ECE 4112 Internetwork Security
Lab 7: Honeypots and Network Monitoring and Forensics

Group Number: _____

Member Names: _____ _____

Date Assigned: March 1, 2012
Date Due: March 8, 2012

Please read the entire lab and any extra materials carefully before starting. Be sure to start early enough so that you will have time to complete the lab. Answer ALL questions and be sure you turn in ALL materials listed in the **Turn-in Checklist** ON or BEFORE the **Date Due**.

Goal: To understand the concept of a Honeypot and how it can prove useful to administrators and network professionals when correctly implemented inside their network topology. Also covered in the lab is he concept of Forensics, which is a way of looking at data you've collected in order to find out what sort of exploit was run on your machine.

Summary: In this lab you will first set up a couple of different honeypots, one on Windows and then one on Linux, to monitor network traffic and look for anything suspicious. You will also use snort to log data and as an Intrusion Detection System. On the forensics side you will examine a few files of captured data from real attacks and see if you can find out what was going on. Finally you will look at a forensic tool and see some of its uses.

Background and Theory:

Honeypot

What is a honeypot? A honeypot is a system whose value lies in being probed, attacked, or otherwise taken advantage of by a blackhat. This idea may sound somewhat counterintuitive at first; why would we want to give one of our valuable systems over to the other side? [1]

The answer to this question depends on what we are trying to accomplish. Spitzner classifies honeypot solutions into two broad categories: production and research. For research purposes, we simply want to collect as much information on our attackers as possible. Production systems are generally used as an added layer of network security. The value of any network security device can be quickly disseminated when one considers the three keys to network security: prevention, detection, and response. Consider "The Burglar Alarm Analogy:" Deadbolting your front door is a way to *prevent*

thieves from entering. A security alarm can *detect* that thieves got past the deadbolt indicating that the preventative measures were not successful. With any luck, your system dials the police who then *respond* by showing up at your house with guns blazing. [1]

A honeypot is the electronic equivalent of an unlocked door, so we can't expect it to add much to the protection layer. It is in the detection of unwanted intruders that a honeypot adds the most value. There is one important reason for this. A honeypot, by definition, should have no legitimate traffic. Consider how much information an IDS system has to sift through, or how many packets are seen by your router or firewall in one day. Entirely too much to sit and watch go by on the wire. If you're hacked, it will be nearly impossible to find the source of the attack with these more conventional network security devices. A honeypot, on the other hand, will collect very little information over the course of a day, but it's likely that *all* of that information has some sort of value to a network administrator. Later in the lab you will see how versatile a honeypot can be in collecting data, whether one is just looking for source and destination ports of scans on their network or want to capture the keystrokes and tools of an attacker for further analysis. [1]

The value added by a honeypot to the response layer of network security (which can be significant) is beyond the scope of this lab. There are numerous whitepapers available on the Internet for those who find themselves interested in the subject. [1] Appropriate websites are listed at the end of this lab.

It's fairly easy to see how a production honeypot might help a company who has been the victim of past attacks, and how a research honeypot might help the security community to analyze the tools and tactics of the blackhat community. What remains to be seen, however, is what sets the two categories apart in terms of the construction, functionality, and topology of the honeypot systems we'd like to deploy. [1]

Traditionally, production honeypots are thought of as simpler and more intuitive than research honeypots, and rightly so. Most network administrators care less about the tactics employed by a hacker and more about the immediate security concerns of their network. A primary goal of a production honeypot, then, is to provide an alert that something might be amiss [1]. This affords system administrators the freedom to select from several commercially available (and sometimes free) honeypot solutions. Examples of such solutions that are currently available include BackOfficerFriendly, Specter, and honeyd.

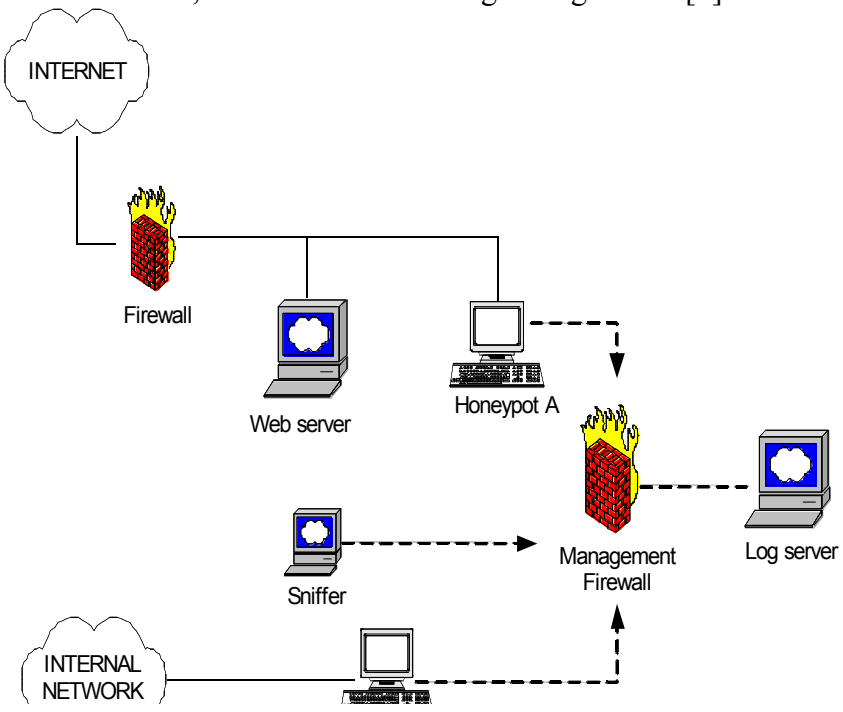
Spitzner classifies these three tools as "low-interaction" honeypots [1]. When we define a honeypot as low interaction, we are referring to how much operability we plan to leave on the victim box for an attacker to play with. Consider the BackOfficerFriendly program, which we will be installing and running for the first exercise in the lab. BackOfficerFriendly can simulate only six services, gathering very little information beyond the source IP and source port that attempted to connect to the ports associated with these services. An attacker cannot break into this system, download his tools, set up an IRC channel, share the secrets of his dark brotherhood, etc. – but this is alright. A low

interaction honeypot like BackOfficerFriendly will still provide an administrator an indication of trouble on his network, and often this is all that's necessary.

Research honeypots, on the other hand, are often homemade solutions that can track an attacker's actions down to the keystroke. Network security professionals and educational institutions often employ research honeypots in the hopes of seeing a hacker in action. A honeypot that is to be used for research will often contain a fully operational operating system running certain services and vulnerabilities. Generally, this type of honeypot is much more difficult to configure and requires more time for upkeep. Also, research honeypots that utilize fully operational operating systems can be dangerous – a skilled attacker is as likely to route successive attacks through our honeypot as any other system that he might compromise [1].

We can see that there are tradeoffs between production and research machines. These tradeoffs can generally be summed up as follows: the more functionality we build into a honeypot, the more useful data we can collect. The more data that we collect, the more likely it is that our honeypot might be used for nefarious purposes. Administrators must be careful when placing a honeypot onto their network. A careful survey of network topology and configuration options on the honeypot itself is imperative. Placing a vulnerable machine in a production network could lead to understandably disastrous results [1].

What can we do when we want to collect a lot of information on a hacker, but are concerned that he might use our honeypot as a platform from which to launch further attacks? Used in conjunction with a firewall and/or router, research honeypots can be made considerably safer. To illustrate this as well as some other points that have already been discussed, consider the following configuration [1]:



The above network might be implemented in either a production or research environment, so it suits our purposes well. Feel free to ignore connections made with a dotted line, i.e. those running to the Management firewall and the associated Log server. These connections are beyond the scope of this lab, and would only be necessary if we were interested in consolidating the logs of multiple honeypots onto a single machine [1].

Honeypot B is placed on the internal network, and as such, had best be a low-interaction "production" honeypot. Assume we have installed BOF onto this machine. We know that if *any* traffic appears on Honeypot B, there are two possibilities: someone on the internal network is doing something they are not supposed to be doing, or our internal network has been somehow compromised by the ever-ominous "Internet Cloud." If someone has slipped through the firewall and is probing our internal network, Honeypot B will most likely know about it – we have configured it to be an attractive target, a networking damsel-in-destress if you will. Detection has occurred, and we are now enlightened and can take appropriate action thanks to Honeypot B's inclusion in the network. If the "sniffer" machine has been placed on the network, it may have collected some data to help us secure things a bit. Then again, the sniffer's data might help us very little or not at all. The important thing is that we have *detected* that some sort of breach has occurred, and now know that some response is called for [1].

Honeypot A is a bit more interesting for our purposes. Placed in the network DMZ (de-militarized zone – typically used in a firewall configuration, such as the one used in the previous lab), this would most likely be configured as a high-interaction honeypot. We will be working with a homemade solution in the lab, so let's think of Honeypot A as such. We might use netcat to monitor port activity with the –l listen option turned on (remember that lab?), in essence giving it the functionality of our BOF honeypot. We can afford to be interested in more than just connection monitoring in this case, though. The DMZ is already fair game to attackers (to a certain extent), so we welcome them in, using the firewall between us and the internet to send any suspicious traffic to Honeypot A. We will see later in the lab how this can be accomplished. We are now faced with two concerns. The first, as mentioned previously, is ensuring the safety of the rest of the Internet community by not allowing the attacker to use our honeypot as a platform from which to launch large-scale attacks, such as Denial Of Service attacks and their various cousins. The second concern is data collection. In a research environment, our goal is to collect as much information and as many blackhat tools as we can [1].

The firewall which sent the traffic to Honeypot A also provides us with a way to limit the danger Honeypot A poses to other machines. In a typical attack, a blackhat will breach the machine, and after securing the box, FTP out to the Internet in order to download rootkits, worms, and various hacker paraphernalia with which he does his deeds. If an attacker is given no limit to the number of outbound connections he can make after compromising the box, fully-functional Honeypot A certainly represents a threat. If we use our firewall to block all outbound connections from the machine, our blackhat will quickly lose interest in Honeypot A and move on, leaving us very little data to analyze. Again we see the correlation between the amount of data that we hope to capture and the amount of risk we are willing to take in capturing it. The more outbound

connections we allow, the more likely that the hacker will stay and play in our honeypot. Deciding on an allowed number of outbound connections is thus left up to the administrator of the given network. Erring on the side of safety, Lance Spitzner would allow around 10 connections. This gives the attacker an opportunity to download his tools and send a few things out over the wire, but will protect against the launch of a significant large scale attack from our machine [1].

We can only learn as much from the blackhat community as we can capture from their activity on our honeypots. Expect that most people breaking into our system are going to make some effort to cover their tracks, and we have a new problem: how are we to collect the appropriate data? To maximize our data intake, we have several options. The firewall is our first point of data collection. We might implement some type of Intrusion Detection System as well. Most often (and in this lab) we will use a monitoring station that is running a packet sniffer such as ethereal or Snort to capture all data sent to Honeypot A. Keep in mind that unless we are under constant attack, we will not be buried in data using this technique. The honeypot ensures that we'll only see "bad" traffic, and will be able to review a relatively small amount of data of relatively high importance [1].

You should now have a fairly decent grasp on what we are trying to accomplish when we use a honeypot to add security to our network. The following exercises will go over how to configure and make good use of several different honeypot techniques. Keep in mind as you work that there are as many good implementations of these machines as you can dream up; this is an extremely flexible technology that has yet to flourish in the world of networking, but is certain to grow as more people realize the honeypot's potential [1].

Aside: Additional information on Honeypots: Client Honeypots

Since normal honeypots depend on external hackers and worms to initiate an attack, they have the advantage that all traffic to them is likely to be attack traffic. For the same reason, they have the disadvantage that they cannot become compromised by attacks that are client-initiated, such as those that occur when browsing the web with a vulnerable version of IE. The counterpart of normal honeypot is a client honeypot, or "honeyclient." Instead of waiting for its server applications or operating system to be exploited, a client honeypot actively crawls the web, searching for servers that will exploit its client applications.

Although the focus is generally on finding web browser vulnerabilities, any client that interacts with the network can be part of client honeypot. In theory, a client honeypot could be trained to discover vulnerabilities not only in web browsers, but in any client application -- for example, ftp, ssh, email, P2P or VoIP clients -- as well as in any programs run as a result of crawling the web (such as MS Word or Adobe Acrobat, which might be run when a doc or pdf file is retrieved).

The information found by a client honeypot can be used in different ways:

- Networks can be informed to correct or shut down web servers containing malicious code (this assumes that the network is itself not malicious)

- New types of client attacks can be discovered and vulnerabilities on the exploited client applications (e.g., IE, MS Word, etc.) can be patched. This can enable zero-day protection (ideally). This is particularly valuable to a company such as Microsoft.
- Blacklists of malicious web sites can be made. They could be used at the network administrator level, or at the DNS level by services such as (possibly) OpenDNS.
- Groups of malicious web sites that redirect users and data among each other can be discovered

A seemingly unavoidable problem of client honeypots is that they are inefficient, in the sense that most of the traffic client honeypots see is benign, since most web sites crawled do not have malicious code. This differs from normal honeypots, which see only suspicious traffic. So some system must be set up on the host to determine whether a system has been compromised as a result of visiting a web site – in effect, this is a host-based intrusion-detection system.

Client honeypots can be designed to simulate a human browsing the web – "driving" a web browser, moving the mouse cursor to click on links, and letting the web browser run the embedded scripts. These are called **high-interaction client honeypots**. Normally, the web browser is run within a virtual machine. When something unusual occurs on the virtual machine, an alert is triggered. Signs that an exploit has occurred are:

- A file was added or changed (e.g. the MD5 hash changed)
- A registry change has occurred
- A new process was started
- Software analysis – e.g., system calls were made that the client application is not known to make

High-interaction client honeypots are great, because they are effective at discovering new client attacks. The problem is that they are expensive to develop and run. VMs use a lot of processing and memory, so a single machine cannot run many VMs. Client attacks can be designed to trigger only when certain user interactions occur – such as doing a particular mouse-over or spending a certain amount of time on a specific web page (a.k.a. a time bomb) – and there may be many possible user interactions for a single web page.

An alternative to emulating all possible interactions with a web site is to instead retrieve web pages using something like "wget" and test the retrieved HTML against known signatures. These are called **low-interaction client honeypots**. It is very fast. Its problem is that, although it can find web sites with known exploits, it is less likely to detect new exploits, which might not match any signatures. It is, however, useful for finding statistics on known exploits (e.g. how many web sites is a particular browser version vulnerable to).

Several software client honeypots have been developed.
High-interaction clients include:

- Honeyclient – IE-based, open source, written in Perl, WinXP/2K, uses MD5 hashes
- Honeymonkey – IE-based, Microsoft uses it in its IE7 Phishing Filter

- UW Spycrawler – Mozilla-based

Low interaction clients include:

- HoneyC - open source, written in Ruby, examines web server's responses for malicious strings using Snort signatures

Client honeypots are a relatively new field of research, and the technique of actively seeking out exploits, rather than passively waiting for them to hit, will be increasingly applied to other areas. An email honeypot system, for example, could publicize email addresses in an effort to attract as much spam as possible to increase the chances of receiving exploits via email. We should expect to see a great deal of work in client honeypots in the future.

Sources for this aside:

- Michael Davis. "Client Honeypots - It's not only the network." Hack In The Box 2006.
- Kathy Wang, MITRE Corporation. "Using Honeyclients to Detect New Attacks." ToorCon 2005. http://www.synacklabs.net/honeyclient/Wang-Honeyclients-RECON.pdf
- Robert Danford, SANS Internet Storm Center. "2nd Generation Honeyclients." http://handlers.dshield.org/rdanford/pub/2nd_generation_honeyclients.ppt
- The Honeyclient Development Project. http://www.honeyclient.org/
- "Strider Honeymonkey Exploit Detection." http://research.microsoft.com/HoneyMonkey
- "Client Honeypot." Wikipedia. http://en.wikipedia.org/wiki/Client_honeypot_/_honeyclient
- Christian Seifert et al. HoneyC - The Low-Interaction Client Honeypot. August 2006. http://www.mcs.vuw.ac.nz/~cseifert/blog/images/seifert-honeyc.pdf

Another Aside: HoneyMonkeys

The HoneyMonkey project was created to help detect attacks on the Internet that exploit browser vulnerabilities and install malware on user's computers without their knowledge. The project was started at Microsoft Research and was created by Yi-Min Wang, Doug Beck, Xuxian Jiang, and Roussi Roussev. The project is sort of "Automated Web Patrol" that makes the web safer by finding vulnerabilities before users have a chance to suffer from web exploits.

HoneyMonkeys are dedicated computers or virtual machines that are specifically set to crawl the web looking to be exploited as opposed to Honeypots that simply wait for someone to come and exploit them. Honey Monkeys are used to emulate a casual web user and see what kind of exploits a normal user encounters by surfing on the Internet. They are usually set to crawl the seedier side of the Internet as that's where many malware is located on the internet; however they also crawl the top million website to make sure that no exploits exist there either.

A HoneyMonkey can crawl about 7,000 websites a day. HoneyMonkeys run Windows XP with different levels of updates to look for sites that exploit specific vulnerabilities. After an exploit is found, follow up work is necessary to determine the cause of the exploit and how it works. From there both the Microsoft Internet Safety Enforcement Team and Microsoft Security Response Center work to prosecute cyber criminals that are responsible for exploits and to make people aware and to fix the vulnerability.

Sources:
http://www.theregister.co.uk/2005/05/17/microsoft_searches_for_web_nasties_with_honey_monkeys/
http://research.microsoft.com/honeymonkey/article.aspx

Intrusion Detection

An intrusion detection system (IDS) monitors network traffic and monitors for suspicious activity and alerts the system or network administrator. In some cases the IDS may also respond to anomalous or malicious traffic by taking action such as blocking the user or source IP address from accessing the network.

IDS come in a variety of "flavors" and approach the goal of detecting suspicious traffic in different ways. There are network based (NIDS) and host based (HIDS) intrusion detection systems. There are IDS that detect based on looking for specific signatures of known threats- similar to the way antivirus software typically detects and protects against malware- and there are IDS that detect based on comparing traffic patterns against a baseline and looking for anomalies. There are IDS that simply monitor and alert, and there are IDS that perform an action or actions in response to a detected threat. We'll cover each of these briefly.

NIDS

Network Intrusion Detection Systems are placed at a strategic point or points within the network to monitor traffic to and from all devices on the network. Ideally you would scan all inbound and outbound traffic; however doing so might create a bottleneck that would impair the overall speed of the network.

HIDS

Host Intrusion Detection Systems are run on individual hosts or devices on the network. A HIDS monitors the inbound and outbound packets from the device only and will alert the user or administrator of suspicious activity is detected

Signature Based

A signature based IDS will monitor packets on the network and compare them against a database of signatures or attributes from known malicious threats. This is similar to the way most antivirus software detect malware. The issue here is that there will be a lag between a new threat being discovered in the wild and the signature for detecting that threat being applied to your IDS. During that lag time your IDS would be unable to detect the new threat.

Anomaly Based

An IDS which is anomaly based will monitor network traffic and compare it against an established baseline. The baseline will identify what is "normal" for that network- what sort of bandwidth is generally used, what protocols are used, what ports and devices generally connect to each other- and alert the administrator or user when traffic is detected which is anomalous, or significantly different, than the baseline.

Passive IDS

A passive IDS simply detects and alerts. When suspicious or malicious traffic is detected an alert is generated and sent to the administrator or user and it is up to them to take action to block the activity or respond in some way.

Reactive IDS

A reactive IDS will not only detect suspicious or malicious traffic and alert the administrator, but will also take pre-defined proactive actions to respond to the threat. Typically this means blocking any further network traffic from the source IP address or user.

One of the most well known and widely used intrusion detection systems is the open source, freely available Snort (http://www.snort.org/) . It is available for a number of platforms and operating systems including both Linux and Windows. Snort has a large and loyal following and there are many resources available on the Internet where you can acquire signatures to implement to detect the latest threats. For other freeware intrusion detection applications you can visit Free Intrusion Detection Software (http://netsecurity.about.com/od/intrusiondetectionid1/a/aafreeids.htm).

There is a fine line between a firewall and an IDS. There is also technology called IPS – Intrusion Prevention System. An IPS is essentially a firewall which combines network-level and application-level filtering with a reactive IDS to proactively protect the network. It seems that as time goes on firewalls, IDS and IPS take on more attributes from each other and blur the line even more.

Essentially, your firewall is your first line of perimeter defense. Best practices recommend that your firewall be explicitly configured to DENY all incoming traffic and that you open up holes where necessary. You may need to open up port 80 to host web sites or port 21 to host an FTP file server. Each of these holes may be necessary from one standpoint, but they also represent possible vectors for malicious traffic to enter your network rather than being blocked by the firewall.

That is where your IDS would come in. Whether you implement a NIDS across the entire network or a HIDS on your specific device, the IDS will monitor the inbound and outbound traffic and identify suspicious or malicious traffic which may have somehow bypassed your firewall or originated from inside your network.

Prelab Questions: None

Lab Scenario:

Make sure windows firewall and realsecure are off from previous lab.

Files needed from the NAS:
On the NAS under "Lab7" are the folders you will need for this lab. Copy the BOF folder (under the Honeypots folder) to your WinXP virtual machine.

Copy the forensics and Honeypots folders to your Virtual Red Hat 7.2 machine. Note this is a big folder and takes some time to copy.

Section 1: Install and run BackOfficerFriendly (A free Honeypot) on your virtual WinXP machine

As described on the BOF source web site: "Known as a "honey pot" for its ability to attract and trap hackers, Back Officer Friendly (BOF) is a popular free download available exclusively from NFR Security, Inc. Back Officer Friendly was originally created to detect when anyone attempts a Back Orifice scan against your computer. It has since evolved to detect attempted connections to other services, such Telnet, FTP, SMTP, POP3 and IMAP2. When BOF receives a connection to one of these services, it will fake replies to the hopeful hacker, wasting the attacker's time, and giving you time to stop them from other mischief".

In the BOF folder you copied to your XP virtual machine, you will find the file nfrbofl.exe. Double click on it then follow instructions for installation clicking yes and OK.

BackOfficerFriendly is now running on your XP virtual machine. There should be a very friendly looking GUI on your screen (if not open it from your system tray). Configuring your BOF honeypot is almost as simple as "installing" it was: in the Options menu, check each service that you would like BOF to emulate on your system. If you check one of the services and see an error message, it is likely that a service is already running on that particular well-known port. You don't have to turn it off; as long as you've got one service available for monitoring you'll be able to see how BOF works. Notice the *Fake Replies* option at the bottom of the menu. Check next to this option as well.

Open a DOS window and run the *netstat –a* command. You should see all of the services that BOF is simulating listed here as *LISTENING*.

Set up telnet to listen for a connection by checking the option in BOF. Use your Red Hat WS 4.0 physical machine and attempt to make a telnet connection to your BOF box on your virtual XP machine. Turn on the *Fake Replies* option.

Q1.1: What happens when a connection is attempted? As a network administrator, why would you want to use BOF on your honeynet?

There are several such honeypots available. The following is taken from http://www.governmentsecurity.org/articles/what-is-a-honeypot.html

Specter
Specter is a commercial product and what I would call another 'low interaction' production honeypot. It is similar to BOF in that it emulates services, but it can emulate a far greater range of services and functionality. In addition, not only can it emulate services, but emulate a variety of operating systems. Similar to BOF, it is easy to implement and low risk. Specter works by installing on a Windows system. The risk is reduced as there is no real operating system for the attacker to interact with. For example, Specter can emulate a webserver or telent server of the operating system of your choice. When an attacker connects, it is then prompted with a http header or login banner. The attacker can then attempt to gather web pages or login to the system. This activity is captured and recorded by Specter, however there is little else the attacker can do. There is no real application for the attacker to interact with, instead just some limited, emulated functionality. Specters value lies in detection. It can quickly and easily determine who is looking for what. As a honeypot, it reduces both false positives and false negatives, simplifying the detection process. Specter also supports a variety of alerting and logging mechanisms. You can see an example of this functionality in a screen shot of Specter.

One of the unique features of Specter is that it also allows for information gathering, or the automated ability to gather more information about the attacker. Some of this information gathering is relatively passive, such as Whois or DNS lookups. However, some of this research is active, such as port scanning the attacker. While this intelligence functionality may be of value, many times you do not want the attacker to know he is being watched. Be careful when implementing any active, automated responses to the attacker.

Honeyd
Created by Niels Provos, Honeyd is an extremely powerful, OpenSource honeypot. Designed to run on Unix systems, it can emulate over 400 different operating systems and thousands of different computers, all at the same time. Honeyd introduces some exciting new features. First, not only does it emulate operating systems at the application level, like Specter, but it also emulates operating systems at the IP stack level. This means when someone Nmaps your honeypot, both the service and IP stack behave as the emulated operating system. Currently no other honeypot has this capability (CyberCop Sting did have this capability, but is no longer available). Second, Honeyd can emulate hundreds if not thousands of different computers all at the same time. While most honeypots can only emulate one computer at any point in time, Honeyd can assume the identity of thousands of different IP addresses. Third, as an OpenSource solution, not only is it free to use, but it will exponentially grow as members of the security community develop and contribute code.

Honeyd is primarily used for detecting attacks. It works by monitoring IP addresses that are unused, that have no system assigned to them. Whenever an attacker attempts to probe or attack a non-existent system, Honeyd, through Arp spoofing, assumes the IP address of the victim and then interacts with the attacker through emulated services. These emulated services are nothing more then scripts that react to predetermined actions. For example, a script can be devloped to behave like a Telnet service for a Cisco router, with the Cisco IOS login interface. Honeyd's emulated services are also OpenSource, so anyone can develop and use their own. The scripts can be written in almost any language, such as shell or Perl. Once connected, the attacker believes they are interacting with a real system. Not only can Honeyd dynamically interact with attackers, but it can detect activity on any port. Most low interaction honeypots are limited to detecting attacks only on the ports that have emulated services listening on. Honeyd is different, it detects and logs connections made to any port, regardless if there is a service listening. The combined capabilities of assuming the identity of non-existent systems and the ability to detect activity on any port gives Honeyd incredible value as a tool to detect unauthorized activity. I highly encourage people to check it out, and if possible to contribute new emulated services.

Mantrap

Produced by Recourse Mantrap is a commercial honeypot. Instead of emulating services, Mantrap creates up to four sub-systems, often called 'jails'. These 'jails' are logically discrete operating systems separated from a master operating system. Security administrators can modify these jails just as they normally would with any operating system, to include installing applications of their choice, such as an Oracle database or Apache webserver. This makes the honeypot far more flexible, as it can do much more. The attacker has a full operating system to interact with, and a variety of applications to attack. All of this activity is then captured and recorded. Not only can we detect port scans and telnet logins, but we can capture rootkits, application level attacks, IRC chat session, and a variety of other threats. However, just as far more can be learned, so to can far more go wrong. Once compromised, the attacker can use that fully functional operating system to attack others. Care must be taken to mitigate the risks. As such, I would categorize this as a mid-high level of interaction. Also, these honeypots can be used as either a production honeypot (used both in detection and reaction) or a research honeypot to learn more about threats. There are limitations to this solution. The biggest one is you are limited to what the vendor supplies you. Currently, Mantrap only exists on Solaris operating system.

Honeynets

Honeynets represent the extreme of research honeypots. They are high interaction honeypots, you can learn a great deal; however, they also have the highest level of risk. Their primary value lies in research, gaining information on threats that exist in the Internet community today. A Honeynet is a network of production systems. Unlike many of the honeypots we have discussed so far, nothing is emulated. Little or no modifications are made to the honeypots. This gives the attackers a full range of systems, applications, and functionality to attack. From this we can learn a great deal, not only about their tools and tactics, but also about their methods of communication, group organization, and motives. However, with this capability comes a great deal of risk. A variety of measures

must be taken to ensure that once compromised, a Honeynet cannot be used to attack others. Honeynets are primarily research honeypots. They could be used as production honeypots, specifically for detection or reaction; however it is most likely not worth the time and effort. Most of the low interaction honeypots we have discussed so far give the same value for detection and reaction, but require less work and have less risk. If you are interested in learning more about Honeynets, you may want to review the book Know Your Enemy.

Section 2: The Homemade Honeypot using Netcat as a Port Sniffer

This homemade honeypot offers us considerably more options than BOF. Our first task, however will be to create BOF-like functionality by using netcat as a port sniffer. Use your Linux 7.2 virtual machine for this honeypot.

The command

#nc –l –p 4000

will listen for any connections made on port 4000. In this way we can detect any suspicious traffic as we would with any port monitor. You can monitor any number of ports in this fashion, making this a much more adaptable low-interaction honeypot solution. There is no pretty GUI though, and much more time is required for upkeep.

What if we changed the above command in the following manner?

#nc –l –p 4000 > worm

Now we are not only monitoring port 4000, we are capturing the data that is sent its way. Herein lies the first new value of the research honeypot. Detection is no longer our primary goal. With this one command we are now able to capture and store automated web attacks. Many worms are spread in just this fashion. Capturing the attack payload may allow us to determine how a worm operates, and can help to stop its spread across the Internet as quickly as possible. Ryan Russell used this method to capture the infamous Code Red II worm in 2001 [1].

Using your physical Linux WS 4.0 machine as a netcat client, you can send data to the listening port 4000 on your honeypot Red Hat 7.2 virtual machine and see that data stored in the file *worm*.

#nc <linux7.2 ip> <port, in this case 4000>
<type anything here>
<CNTL C to quit>

Q2.1: Now look at the worm file on the Linux 7.2 machine, what do you see here?

There are other ways to adapt homemade port monitoring to be even more versatile. The PERL scripting language is often used to generate fake server responses that are sent to the attacker when he connects to a particular service. Learning PERL is not necessary to understand the concept of honeypots, however, we will not be asking you to do anything like "write a login script that would run when a telnet connection is made to the honeypot"; just know that such capabilities exist, and have been used to great effect in collecting information on attacks.

Section 3: Set up and use Ethereal to capture packets

Part 3A:

1. Ethereal is already installed on your Red Hat 7.2 virtual machine. Start ethereal on the 7.2 virtual by
Click on the FOOT→ Programs → Internet → Ethereal
2. To begin packet capture go to Capture → Start
3. Check the settings to ensure Ethereal is listening on the proper interface and that it is in promiscuous mode
4. Notice the filter rule box to enable filtering of observed traffic

Ensure that telnet is enabled on both your physical Red Hat WS 4.0 machine (client) and your Red Hat 7.2 virtual machine (server).
On your 7.2 virtual machine, create a user other than root if you do not still have one from a previous lab.

#adduser user1
#passwd user1

5. Establish a telnet connection from your WS 4.0 physical machine to you 7.2 virtual machine, and produce a small amount of traffic so it can be observed in the Ethereal log. You cannot telnet from the Red Hat WS 4.0 physical machine to the Red Hat 7.2 virtual machine using root so use your new user as the account you telnet to.

Q3.1: What packets did you observe?
Attachment 1: Submit screen shot of packets.
Q3.2: Describe the content of the packets.

Part 3B:
Run the old imapd exploit we saw in a previous lab (For more detailed information on how to do the imapd attack see Appendix A.)

1. Start Ethereal and begin packet capture mode on the Red Hat 7.2 box (imapd server).
2. Execute the imapd-ex from the attacker machine (RH WS 4.0) against the imapd server machine as done in the buffer overflow lab while Ethereal is running in packet capture mode (do not set any filters) on the imapd server machine. Do this by looking in the root directory for the executable file imapd-ex. From that directory execute:

```
#(./imapd-ex 3000 ; cat) | nc a.b.c.d 143
```

Q3.3: What packets did you observe?
Attachment 2: Submit screen shot of packets
Q3.4: Describe the content of the packets.

Section 4: Set up and use Snort to capture packets

(http://www.snort.org/)

Let's consider how we might capture a detailed record of the packets sent to and from our honeypot using a sniffer. Our first task will be to install Snort (onto your Linux 7.2 machine), a sniffer similar to Ethereal. Snort is a very versatile tool that can operate in 3 modes: 1. Sniffer Mode 2. Packet Logger Mode 3. Network Intrusion Detection Mode. We will be examining the first 2 modes now and the third mode in Section 6. Snort requires the presence of *libpcap* on your system to monitor packets, so we will first be installing that program.

(For those who are interested, Snort is also available for Windows at: http://www.snort.org/downloads. Note: you would also need to install WinPcap for Snort to work. WinPcap may be found at http://www.winpcap.org/)

You should have already copied snort and libpcap files from the Honeypots folder on the NAS onto your Linux 7.2 machine. Unzip each file with *tar xvfz* and run the following commands in their respective directories:

#./configure
#make
#make install

After this is done for both programs, you should be ready to capture packets off the wire. Start Snort by using the following command in the *snort-1.9.1/src* directory:

#./snort -vd

The –v option shows the IP and TCP/UDP/ICMP headers. The –d option tells snort to display the packet data as well.

Use your Linux WS 4.0 machine to send some network traffic to the Linux 7.2 box. It doesn't matter how you do so, just send something. You will see Snort displaying the activity to standard out on the honeypot's screen. Unencrypted data appears (in a somewhat chopped up form, admittedly) in Snort's output.

This data doesn't do very much good if it's just scrolling past us. We need a way to log the data, so we can peruse it at our convenience. Luckily, Snort has a built-in Packet Logger Mode. In your *snort-1.9.1/src* directory, create a subdirectory called *log*. Run the command:

#./snort –vd –l ./log

Once again, direct some traffic at your honeypot. Also, send some traffic out from the honeypot.

Q4.1: Explain how the *–l* option organizes the logging of network traffic in your new *log* directory.

Most often we would want to use a separate machine to do our Sniffing duties. Storing the logs locally will only lead to their quick deletion at the hands of an attacker with any skill.
The following is taken from the Snort User's Manual at http://www.snort.org

If you're on a high speed network or you want to log the packets into a more compact form for later analysis, you should consider logging in binary mode. Binary mode logs the packets in tcpdump format to a single binary file in the logging directory:

```
./snort -l ./log -b
```

Note the command line changes here. We don't need to specify a home network any longer because binary mode logs everything into a single file, which eliminates the need to tell it how to format the output directory structure. Additionally, you don't need to run in verbose mode or specify the -d or -e switches because in binary mode the entire packet is logged, not just sections of it. All you really need to do to place Snort into logger mode is to specify a logging directory at the command line using the -l switch--the -b binary logging switch merely provides a modifier that tells Snort to log the packets in something other than the default output format of plain ASCII text.

Once the packets have been logged to the binary file, you can read the packets back out of the file with any sniffer that supports the tcpdump binary format (such as tcpdump or Ethereal). Snort can also read the packets back by using the -r switch, which puts it into playback mode. Packets from any tcpdump formatted file can be processed through Snort in any of its run modes. For example, if you wanted to run a binary log file through Snort in sniffer mode to dump the packets to the screen, you can try something like this:

```
./snort -dv -r packet.log
```

You can manipulate the data in the file in a number of ways through Snort's packet logging and intrusion detection modes, as well as with the BPF interface that's available from the command line. For example, if you only wanted to see the ICMP packets from the log file, simply specify a BPF filter at the command line and Snort will only see the ICMP packets in the file:

```
./snort -dvr packet.log icmp
```

Section 5: Scan of the Month Challenge

The Honeynet Project is made up of a group of network security professionals who have taken it upon themselves to investigate the motives, tools, and tactics of the blackhat community by utilizing various honeypot technologies. Their website,

www.honeynet.org, contains links to some interesting exercises that they call Honeynet Challenges, allowing budding network aficionados to analyze different attacks that have been captured on Project honeynets. We will be considering Scan-of-the-Month 22 (SOTM 22) for this assignment.

You should have already copied the SOTM html files and Snort logfile from the NAS to the virtual Red Hat 7.2 Linux machine onto which you earlier in this lab installed Snort. Unzip the logfile in the directory honeypots_sotm22 directory, using the *gunzip* command. You should also have a binary file "decoder", a file "foo.c" and a file "nvpdata.txt".

Snort allows us to use several command line options to review data collected in the logfile. The entire file is very large and unwieldy, so in order to analyze the data it contains it is necessary to look for specific packets. If for instance, you have determined the attacker's IP address, and wish to view only the packets sent by the attacker, you can do so with the following command:

#snort –vdr <logfile> host <attacker's IP address> | more

Alternately, you could redirect this command's output to a file for a more detailed analysis:
#snort –vdr <logfile> host <attacker's IP address> > attack.txt

The binary logfile can also be run through a Snort config file to extract certain data according to rules set in the config file, using a command such as:

#snort –r <logfile> -c <config file> -l ./log

This command will dump the ASCII payload into text files that are much more readable than typical Snort output.

Use Snort's help mechanism to view many other options that can be used on the command line. Find one that looks interesting and use it to extract some data from the logfile. You can also open the unzipped log file in ethereal, but it takes about 5 minutes to load.

Now on to the Challenge. Take a look at the Scan 22 html file that you copied to your machine. There is an introduction to the challenge, as well as a set of questions to be answered concerning the nature of the attack. Using the Snort log file as well as the extraction techniques described above, it is possible to determine answers to each of these questions.

Note that full analysis of one of these attacks takes quite a bit of skill. Never fear, we have included a detailed analysis submitted by Eloy Paris to the Honeynet Project when the scan was first released. Do your best to answer the questions yourself. Hints have been provided here for you, and not all of the questions posed in the challenge are repeated here, so be sure to take a close look at Eloy Paris' solution to understand just

how much skill and knowledge are sometimes required to discover the motives and techniques of a hacker.

In your answers, include responses to the questions you felt you could answer. If your answers are a little different than those in the analysis, it is OK. For questions you were unable to answer, describe what methods you used, what background information would have been most helpful in the analysis, and whether looking at the provided solutions illuminated the situation or not.

The Challenge:
After penetrating the Linux system using the WU-FTPD vulnerability, the attacker deployed a backdoor binary and then proceeded to use the system for a certain nefarious activity. Your mission, should you choose to accept it, is to determine what the activity was and how it was accomplished. All the necessary evidence is contained in a snort binary capture file. The IP address of the honeypot is 172.16.183.2. Using the snort binary capture answer the following questions. The hacker is using an NVP backdoor tool, for information on this, see the following link:

http://www.honeynet.org/reverse/results/project/020601-Analysis-IP-Proto11-Backdoor.pdf

A few key pages from this document have been copied into Appendix B. These pages explain how data is entered in the packets, that is, where the command is and parameters such as IP addresses. However, we can not simply read the provided log file because the packets were encrypted. Luckily, the SCOM provided us with an executable file, "decoder" that will process the data. This can be run by typing:

#chmod u+x decoder

#./decoder –p <snort logfile>

while in the same folder as decoder. This processes every packet and decodes those that are nvp protocol. This program has already been run for you and the data put into the file nvpdata.txt. Because many of the packets were not nvp, there were many, many "reading packet…" lines that were unnecessary, so many of them have been discarded, there are also a couple of comments added. This file along with Appendix B will help you greatly in answering some of the following questions.

Q5.1: What is the attacker's IP address? (Hint: You will need to convert from Hex to decimal)

Q5.2: What is the attacker doing first? What do you think is his/her motivation for doing this?

Q5.3: What is the purpose of 'foo'? Can you provide more insights about the internal workings of 'foo'? Do you think that 'foo' was coded by a good

programmer or by an amateur? (Note: Eloy Paris' decompiled foo.c has been included for you.)

Q5.4: What is the purpose of './ttserve ; rm -rf /tmp/ttserve' as done by the attacker?

Q5.5: How do you think the attacker will use the results of his activity involving 'foo'?

Q5.6: If you administer a network, would you have caught such NVP backdoor communication? If yes, how? If you do not administer a network, what do you think is the best way to prevent such communication from happening and/or detect it?

Now this lab will shift to the more forensics side of network monitoring. Like looking at a crime scene after the fact, we rely on data collection to see what happened, but we need to know what is useful data and what is not. Forensics in network security is a way of finding out what has happened, if anything, to our machine so we best know how to fix them. This can be done by looking at the packets directly, to see if we notice any familiar exploits, or by looking at our machine to see if anything has been changed (in this lab, by using Aide).

Section 6: Using SNORT to act as an IDS (Intrusion Detection System)

Besides just having snort watch and/or record all of the packets going by, we can set it up to only look at certain ones, either by type, source IP (with our without port), destination ip (with or without port) or a combination of these. We can then make specific alerts and actions for each potential threat. This is done by creating rules that we will then tell Snort to use when it is running. The rules are kept in *snort-1.9.1/rules* directory. Snort also has a config file, called snort.conf, which we tell it to use by adding:

#snort -c /<wherever your snort folder is>/snort-1.9.1/etc/snort.conf

to the command line when running snort. They are command line and configuration file based options to control how snort runs. But the rules which snort uses to match attack patterns are always taken from configuration files.

Just running snort with rules, as done above, will log only the packets that have signatures that match one of the rules, they will be logged in */var/log/snort*. You can also output the alerts to the screen with

#snort –A console –c /<wherever your snort folder is>/snort-1.9.1/etc/snort.conf

Below is some information on the snort.conf file and how to make rules. Use this info and what you observed about the buffer overflow packets in exercise 3B to answer the questions in this part of the lab.

Configuring Snort:

Snort can be configured to watch incoming network traffic directed to the host that it is running on or to scan all network traffic that the host "sees." This option is configured using the configuration variable `HOME_NET` in the `snort.conf` file. There are several different ways one can set this up. Here are four different choices.

1. Monitor one host
   ```
   var HOME_NET 10.1.1.100
   ```
2. Use the default Ethernet interface's IP address
   ```
   var HOME_NET $eth0_ADDRESS
   ```
3. Monitor a range of hosts
   ```
   var HOME_NET 10.1.1.0/24
   ```
4. Monitor hosts on different subnets
   ```
   var HOME_NET [10.1.1.0/24,192.168.1.0/24]
   ```

There are a number of administrator configurable variables that snort rules use. By default these rules use `HOME_NET` as the target address(es). Each of these addresses can be set using the same type of strings as `HOME_NET`. Take care not to use any space characters when constructing these address lists.

Content inspection of packets for each rule that snort uses can be very costly. You are advised to reduce the number of rules to improve overall performance. Attackers could use Snort's rules to mask their attacks in a sea of alerts.

Snort also comes with some interesting support modules. Among them portscan, portscan2, fragmented packet and the stream4 preprocessor stand out. The fragmented packet preprocessor allows snort to reassemble (intentionally) fragmented packets. The portscan and portscan2 preprocessors handle port-scanning techniques. Portscan2 is preferred as it is more general than portscan.

The stream4 preprocessor allows for stateful inspection of TCP traffic. This is important to counter several IDS defeating techniques. Attackers often will send packets out of sequence or retransmit different byte ranges of data in order to evade detection by an IDS. Tools like snot will easily overload Snort with bogus alerts cloaking and attacker's true attack. Snot is not used in the lab because it needs to be modified to use the newest version of libnet.

Writing Snort Rules:

```
ACTION PROTOCOL IP[/mask] PORT -> IP[/mask] PORT (OPTIONS)
```

ACTIONS

1. alert - log packet info and alert message to the alert file

2. log - log this packet without sending out an alert
3. pass – skip matching other rules

PROTOCOL

```
TCP, UDP, BGP, ARP, IP, ICMP, etc.
```

IP

1. A specific address
   `10.1.1.101`
2. A range of addresses using CIDR notation mask
   `10.1.1.0/24 or 10.1.0.0/16`
3. List of comma separated addresses and ranges
   `10.1.1.101,192.168.1.0/24`
4. Any address using the `any` keyword
5. Using the `$HOME_NET` variable which specifies the home network
6. Use an exclamation mark, `!`, for address (range) negation.
   `!10.1.1.101`

PORT

1. A specific port
   `22`
2. A range of ports
   `3000:6000`
3. Port ranges lower than (example all privileged ports)
   `:1024`
4. Port ranges above (example all non-privileged ports)
   `1024:`
5. Any port using keyword `any`
6. Use an exclamation mark, `!`, for port (range) negation.
   `!22`

OPTIONS

The option field is enclosed by parenthesis using a semi-colon as an option delimiter. Snort will ignore white space in the options allowing users to create easy to read option rules.

```
( option )
```

```
( option : option )
```

Option rules are applied to perform some action or select packet with finer granularity than the generic snort rule. There are a number of protocol specific options, but those will not be covered. Those require more knowledge about the different protocols. You can read about them in the referenced material.

1. Messages to write in the logs or alert files. Messages generally contain the name of the exploit defined by Mitre's Common Vulnerabilities and Exploits at http://www.cve.mitre.org/, or by CERT's Advisory Network at http://www.cert.org/advisories/. Good messages also directly link to the exploit definitions.

   ```
   (msg:"TEXT")
   ```

2. Logging to special log files. This allows you to simplify sifting through your log files instead of using on big alert file.

   ```
   (logto:"FileName")
   ```

3. Packet payload scanning. This technique is used to identify sequences of known patterns in attacks. To improve legibility, snort does not consider white space as part of the rule. Instead rules are written using a combination of ASCII and hex codes. The pipe character, |, is use to identify variable portions of the payload pattern. You may also specify more than one pattern to match. Keep in mind this is an expensive operation. Construct your rules wisely.

   ```
   (content: "CONTENT STRING")
   ```

Q6.1: Write a rule to detect the imapd-ex attack.

Add the rule to the SNORT rule base on the RH 7.2 server box

- Create your own rules file in the rules directory of */usr/local/etc/snort/rules*
- Modify */<wherever your snort folder is>/snort-1.9.1/etc/snort.conf* to recognize your new rules file

Run SNORT in IDS mode on the Red Hat 7.2 server box

- Execute the imapd-ex attack from the attacker box (RH WS 4.0) again

Q6.2: Did SNORT give an alert?
Attachment 3: Submit a screenshot or output of alert

Section 7: Advanced uses of Ethereal

Exercise 7 includes analysis of data sets collected from a real honeynet run by Mr. John Levine. Because this is a honeynet, there should be no traffic on it, since it is just sitting there. That means that any outside traffic is immediately suspicious. See what these outsiders try to do, either commands they try to run or files they try to move.

Conduct basic forensics on a honeynet log.
Use an analysis tool of your choice to determine the following information about the attacks represented in the file *snort-0320@0001.txt.*

Q7.1: Specify any hacking activities from the data sets.
List details such as Time of attack (start, end and duration), Source IPs, Target IPs, Hacker's activities and Service of the target

Q7.2: Suggest possible security measures to prevent these attacks. What can be done with the information gathered in order to enhance security of the network (think firewall and IDS)?

Q7.3: Use an analysis tool of your choice to determine the following information about the attacks represented in the files *snort-0920@0001.log* and *snort-0926@1541.log*
You can do either one of them.
Specify any hacking activities from the data sets.
List details such as
- **Time of attack (start, end and duration)**
- **Source IPs (IP addresses and ports)**
- **Target IPs (IP addresses and ports)**
- **Hacker's activities**
- **Hacker's objectives (were they accomplished?)**
- **OS and service of the target**

Suggest possible security measures to prevent these attacks. What can be done with the information gathered in order to enhance security of the network (think firewall and IDS)?

Aside: Ethereal Filtering

Overview
Ethereal is used by network professionals around the world for troubleshooting, analysis, software, and protocol development. It has lots of features that you expect to find in a

protocol analyzer; however, its easiest to use and most beneficial feature is the ability to filter large amounts of packet data. By filtering the data, you can look for very specific information from TCP communications, to hacking attempts, to ARP requests.

The Big Picture
Filtering Ethereal data provides several perks to make your packet browsing easier to handle and a more enjoyable experience. It is easy to use when you need quick results, but can be extremely powerful when necessary. Additionally, through filters more time can be spent looking at packet payloads than simply looking for the right stream.

The Problem
Capturing data with Ethereal can often return hundreds, if not thousands, of packets. Looking at these packets one-by-one can be a very tedious and exhausting process. Additionally, if you simply want to look at the communications between one or two machines, all of the excess packets simply make the process more difficult.

How it Works

All filter statements should be entered into the filter bar at the top of the Ethereal interface. Valid statements will appear in green while invalid statements will appear in red. Additionally, you may click on the filter icon and use the Ethereal Filtering GUI to create filtering statements. Regardless, to get started with filtering, first capture some data.

Ethereal uses many basic statements to filter. You can create everything from regular expressions to compound statements. In general, you will have some reserved Ethereal word followed by a == or != parameter, and then what you are searching for. Here are several examples:

Examples

Only show datagram packets:
eth.type == 0x800

Only show ARP communication to a certain host:
eth.dest eq 21:b7:ab:34:6b:23
Only show certain packet sizes:
Frame.pkt_ln > 1200

Only show a specific conversation using the AND operator:
(ip.src == 192.168.1.10) && (ip.dst == 192.168.1.11)

Only show HTML HEAD packets:
http.request.method == “HEAD”

Only show AIM conversation information:
aim.icbm.extended_data.message.text = "…"

Only show MX record DNS lookups:
dns.qry.type == "MX"

Only show certain SSH types:
ssh.protocol == 1

References and Images
Ethereal
http://www.ethereal.com/

Ethereal (software). (2006, October 15). In Wikipedia, The Free Encyclopedia. Retrieved 14:30, October 25, 2006, from http://en.wikipedia.org/w/index.php?title=Ethereal_%28software%29&oldid=81683912

Section 8: Introduction to AIDE (Advanced Intrusion Detection Environment)

Reference: http://www.cs.tut.fi/~rammer/aide.html

AIDE is a tool for trying to detect if someone has been on our machine and changed anything. If we know or suspect that someone has been on our machine, we can run aide to see what files have been modified, this will be a great help as we try to see what someone has done. AIDE works by creating a checksum of files in specific, user defined directories, saving these in a database, and checking them against the same files at a later date. The major drawback to AIDE is human, that is, it has to be run BEFORE an attack or it is worthless.

Note: The method for installing aide listed here is NOT standard. There are some issues with how aide's build script runs. The aide mailing list (and John Levine) served as the source for the necessary modifications. Also, downloading the latest and greatest version of the libmhash source resulted in compilation problems. For future reference, should your installation not locate libmhash, choose an older version.

Note: You must be root or use sudo to install these binaries.
Copy aide-0.9.tar.gz, mhash-0.8.16.tar.gz, and aide.conf from the Forensics folder to your */home/tools* directory on your Linux 7.2 virtual machine.
Install Aide on your Linux 7.2 machine by following the directions below.
Install libmhash:
#tar xfz mhash-0.8.16.tar.gz
#cd mhash-0.8.16

#./configure
#make
#make check
#make install

Install aide:

1. Extract the source code
   #tar xfz aide-0.9.tar.gz
2. Installation

   #./configure –-with-mhash
   #make
   #make install
3. New Config. File
   Copy aide.conf over the /usr/local/etc/aide.conf file. This file tells aide which directories to make checksums of.

Lab assignment:

1. First, we need to run AIDE initially to set up the data base.
   #aide --init (note: 2 dashes)
   This creates the file aide.db.new in /usr/local/etc. Before we can run aide against this file (which is our checksums of uncorrupted files) we need to rename it aide.db, so in /usr/local/etc:
   #mv aide.db.new aide.db
   This needs to be done everytime aide --init is run

2. Now, we want to test aide.
   #aide --check (note: 2 dashes)
   Ignore the line: "Not implemented in db_readline_file 310"
   Add a new user on the Red Hat 7.2 Virtual Machine and test aide again.
   Copy the lrk4 login file over */bin/login* (after backing up the original) and run the aide check again.

Q8.1: What does Aide show after each of the changes (new user and new binary file)?

Q8.2: Where is a good place to keep a clean copy of Aide and the database? Why?

Section 9: Snare for Windows

(http://www.intersectalliance.com/projects/SnareWindows/)

We have seen how you can use Snort to look at customized alerts in Linux. What if we wanted the same ability in Windows? This portion of the lab will guide you to set up similar functionality in Windows using a tool called Snare.

1. Download the file snaresetup-2.4.3.exe from the NAS
2. Double click to install (use all default options)

a. Notice that Snare will ask to gain control of your windows system logs. This is OK.

3. Once the program has been installed, go to START>> InterSectAlliance >> Audit-configuration. (unless the program has automatically started).

Try opening a command window (START >> RUN >> type in "cmd"). Notice that the events populate automatically. (You may have to restart audit (activity menu).

Q9.1 Identify some of the events that have already appeared. Why might this be important with regard to safety?

Attachment 4: Take a screenshot of the command window starting and point out the specific event

Since Snare has taken control of the windows systems logs, it is already monitoring your logs. The benefit of Snare is that it can be setup so that a system administrator can remotely pull the logs and set alerts. Let's make use of some of its functionality.

With the program started, go to SETUP >> REMOTE CONTROL CONFIGURATION

1. Select the option for "allow snare agent to be remotely controlled"
2. Select "Allow remote control only from this IP Address". Put in your RH WS 4.0 IP
3. You can optionally set a password
4. We will leave the DEFAULT PORT as is (6161)

You may have to do an Activity=> Apply&Restart Audit here.

From your Red Hat WS 4.0 machine, open a web browser and go to the URL: http://a.b.c.d:6161 where a.b.c.d is the IP address of your WinXP machine.

Q9.2 How could this remote control functionality be useful?

Section 10: Forensics Investigation with the Penguin Sleuth Kit

We are going to use the Penguin Sleuth Kit, to simulate what one would do after a successful attack was performed on one of our systems. The Penguin Sleuth Kit is a bootable Linux distribution based on KNOPPIX. It is available from http://www.linux-forensics.com/downloads.html and has some "tremendously useful" tools, including The Coroner's Toolkit (TCT), Autopsy, and The Sleuth Kit, as well as penetration testing and virus scanning tools. We will be using a bootable CD to do a "postmortem" analysis of one of our virtual machines. But first, we must make a backup copy of our Red Hat 7.2 virtual machine to use. If you already have a copy made from one of the earlier labs, great! We will use that.

First, boot your RedHat7.2Copy machine and log in as root. Bring up a terminal and cd to your /bin directory. We are about to do something evil to make our machine unusable (such as what an attacker might do):

#cp login login.bak
#rm login

Answer yes if it asks you if you are sure you want to remove 'login.'

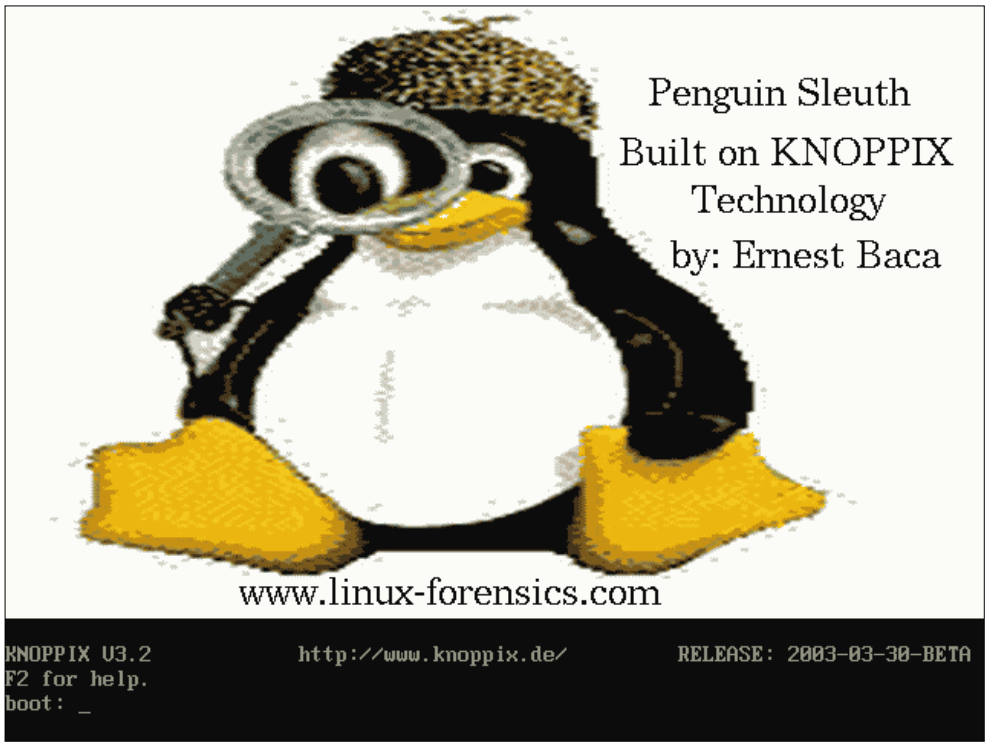
Oh, no! We just removed our login program! What will we do? If this were a big attack on an enterprise system, we would immediately pull the plug from the wall, to avoid losing any evidence. We will simulate this by going to the Power menu in VMWare and choosing Power Off.

Okay, now we have a messed up system. If this were an enterprise-level system that had been attacked, we would immediately inform the proper authorities, who would then come watch us (or do so themselves) remove the hard drive, do a bit-by-bit copy of the drive onto a spare drive or image file, then store the drive away in a safe place. We (or the FBI computer crimes division) would then use the copied image to do a forensic analysis of the "crime scene."

In our case, however, we will not make a backup image, but instead use The Penguin Sleuth Kit to restore our system to a usable state. Download the file penguinsleuth-07-05-2003.iso into your tools directory. In VMWare, edit your RH7.2Copy's Virtual Machine Settings. Choose Add… under Hardware, DVD/CDROM Drive, Use ISO Image, and point it to the ISO you just downloaded.

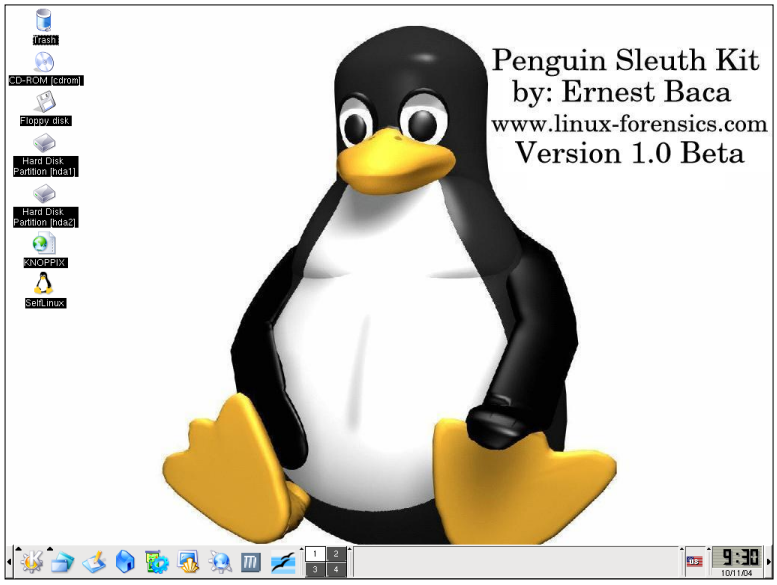
Restart your Virtual Machine, but this time make sure your hit F2 first! (You may have to click on the VM to switch focus to that machine). We need to enter our BIOS to boot off our "CD-ROM." If you can't hit F2 fast enough, turn off the VM and try again. Go to the Boot menu in the BIOS, choose CD-ROM Drive, and hit the + key to move it above the Hard Drive. Go to the Exit menu and choose Exit Saving Changes (answer Yes).

Your VM should now boot into the Penguin Sleuth Kit boot screen:

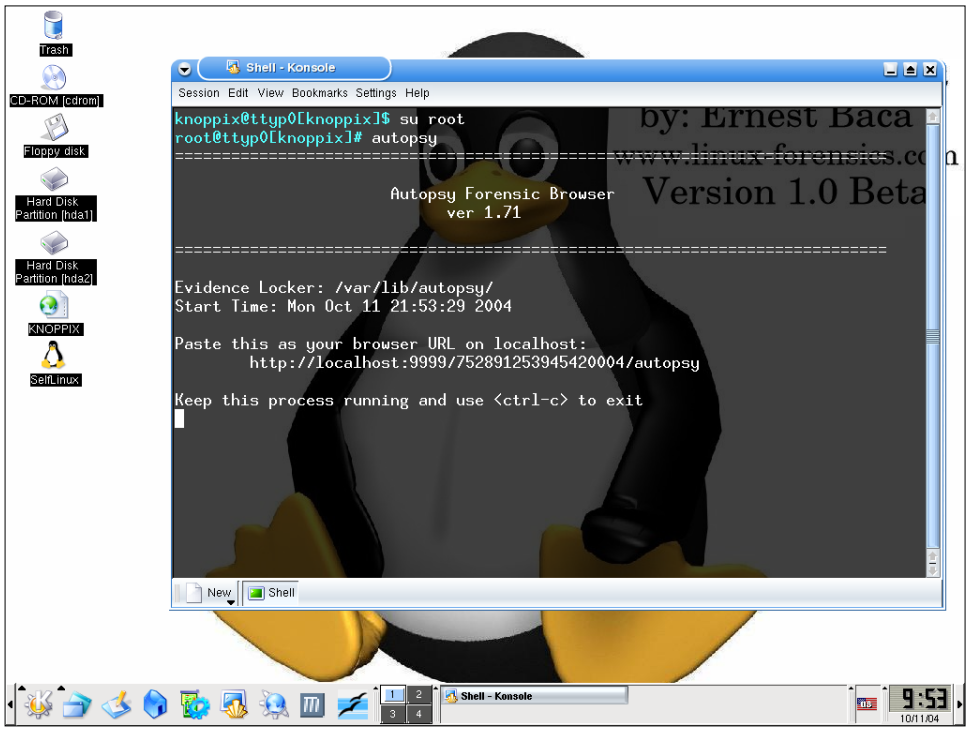


Hit enter to boot.

Once X starts, you will see a screen that looks like this:



Now we will start the Forensics investigation. Open K-Menu--> System-->Konsole. Type `su root` (you will not have to enter a password because root's password is not set yet).



Now we will start Autopsy, a powerful tool to analyze our hard drive image. Type `autopsy` to start it. Take note of the URL displayed after autopsy starts. Now open mozilla by clicking K-Menu-->Internet-->"Mozilla Components"-->"Mozilla Browser", and type in the URL specified after autopsy started up. You should see a friendly GUI that looks like this:

Click Open Case, New Case. Name the case whatever you want, and then put your names in the Investigator Logins. Then click New Case, OK, OK, Add Host. Name the host whatever you want (e.g. `rh72copy`). Under timezone put `GMT-0500`. Then click

Add Host, OK, OK. Select an investigator (whoever is at the keyboard) and click OK, then Add Image. The image location is `/dev/sda2`. (In a real forensics investigation, we would use the image file we acquired from our victim hard drive.) Leave the Symlink option selected; the file system type is `linux-ext3`. The mount point will be `/mnt/sda2`. Choose Ignore for the MD5 for now (though in a real investigation, this would be key to guaranteeing that the image did not change in any way). Then click Add Image, OK.

Now we will look at a few things on our file system. Note that this analysis will be completely passive; the file system is not even mounted! Choose OK to open our image, and then choose File Analysis to look at our files. Under view directory, type `bin` and click OK. If it messes up, try clicking OK again. You should now see the contents of the /bin directory on the hard drive, including modified, accessed, and changed times, as well as size, UID, GID, and meta data inode link. Scroll down to where you can see login and login.bak. Notice how login is flagged as being deleted.

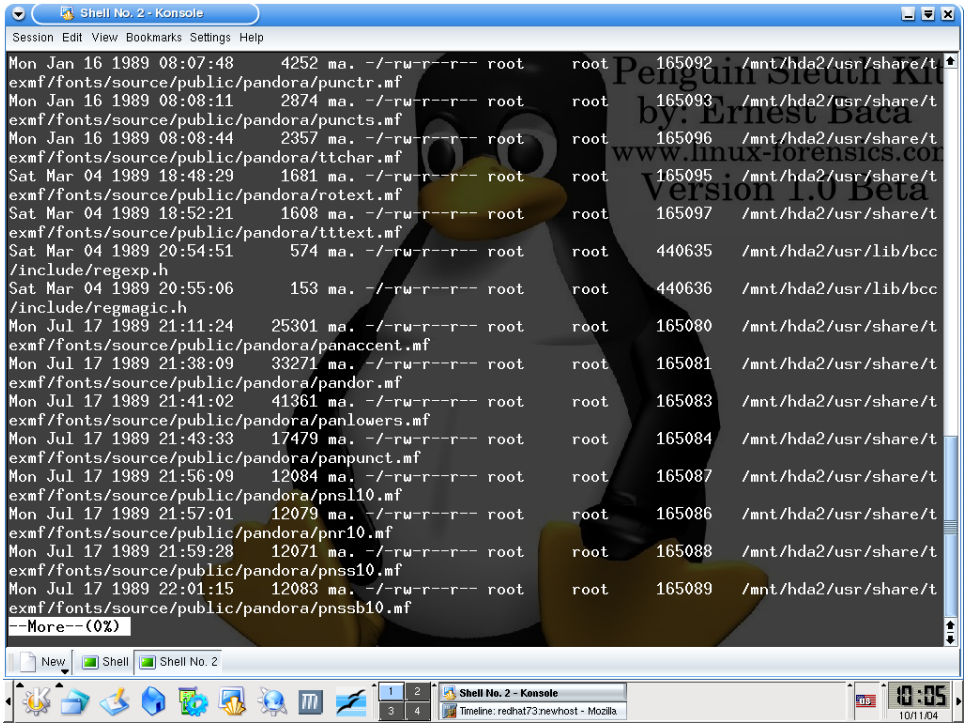
One of the most powerful aspects of Autopsy is its ability to show an investigator a time line of what happened on a system. Click Close in the top frame, and choose File Activity Time Lines at the menu. Choose Create Data File at the top, and select the check box next to /mnt/sda2, and click OK. (This may take a while.) Click OK when it completes. At the next page (Create Time line page), specify a starting date one day in the past, and specify an ending date one day in the future. Enter whatever you want for the filename, and choose sda2 as the image that contains /etc/passwd and /etc/group. Then click OK. (It may complain about not finding the /etc/passwd and /etc/group files. This is okay; see below.)

When the time line is created, don't click OK. That would display the time line in the browser, crashing it (it's big). Instead, follow the recommendation and look at it in a text editor. Hop down to a shell prompt (Alt+Tab) and

#cd /var/lib/autopsy/<casename>/<hostname>/output
#less <outputfilename>

You will now be perusing a time line of all the file activity on the hard drive, complete with timestamp, file size, modified/accessed/created flags, permissions, UID, GID, inode, and filename. It should look something like this:

You can hit Q to get out of less.


Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help
Mon Jan 16 1989 08:07:48 4252 ma. -/-rw-r--r-- root root 165092 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/punctr.mf
Mon Jan 16 1989 08:08:11 2874 ma. -/-rw-r--r-- root root 165093 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/puncts.mf
Mon Jan 16 1989 08:08:44 2357 ma. -/-rw-r--r-- root root 165096 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/ttchar.mf
Sat Mar 04 1989 18:48:29 1681 ma. -/-rw-r--r-- root root 165095 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/rotext.mf
Sat Mar 04 1989 18:52:21 1608 ma. -/-rw-r--r-- root root 165097 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/tttext.mf
Sat Mar 04 1989 20:54:51 574 ma. -/-rw-r--r-- root root 440635 /mnt/hda2/usr/lib/bcc
/include/regexp.h
Sat Mar 04 1989 20:55:06 153 ma. -/-rw-r--r-- root root 440636 /mnt/hda2/usr/lib/bcc
/include/regmagic.h
Mon Jul 17 1989 21:11:24 25301 ma. -/-rw-r--r-- root root 165080 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/panaccent.mf
Mon Jul 17 1989 21:38:09 33271 ma. -/-rw-r--r-- root root 165081 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/pandor.mf
Mon Jul 17 1989 21:41:02 41361 ma. -/-rw-r--r-- root root 165083 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/panlowers.mf
Mon Jul 17 1989 21:43:33 17479 ma. -/-rw-r--r-- root root 165084 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/panpunct.mf
Mon Jul 17 1989 21:56:09 12084 ma. -/-rw-r--r-- root root 165087 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/pnsl10.mf
Mon Jul 17 1989 21:57:01 12079 ma. -/-rw-r--r-- root root 165086 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/pnr10.mf
Mon Jul 17 1989 21:59:28 12071 ma. -/-rw-r--r-- root root 165088 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/pnss10.mf
Mon Jul 17 1989 22:01:15 12083 ma. -/-rw-r--r-- root root 165089 /mnt/hda2/usr/share/t
exmf/fonts/source/public/pandora/pnssb10.mf
--More--(0%)
Penguin Sleuth Kit
by: Ernest Baca
www.linux-forensics.com
Version 1.0 Beta
New
Shell
Shell No. 2
Shell No. 2 - Konsole
Timeline: redhat73:newhost - Mozilla
10:05
10/11/04

Q9.1: What files did the user "rpcuser" access? (Hint: grep for " rpcuser "; if you couldn't access /etc/passwd earlier, grep for " 29 " (rpcuser's UID))

As you can see, Autopsy is very powerful, as it lets you see what happened at a very low level. Now go back to the browser and choose Close from the top frame. Then click Close Host, Close Case, Main Menu. Now close the browser.

Now we should put our machine back in working order. To do this, we will actually modify the file system on the hard drive (which one would not do during a forensic investigation). Jump to the shell prompt and do:

#mount /dev/sda2
#cd /mnt/sda2/bin
#mv login.bak login

Done! We've restored our system to working order. Now:

#cd ../..
#umount sda2

Right Click on the desktop and choose Shutdown.
Go back to Edit Virtual Machine setting and for the CDROM connection, click on "Use a physical drive".

References

Books:

[1] *Honeypots: Tracking Hackers* by Lance Spitzner. Addison Wesley 2003.

Websites:

http://www.honeypots.net/

A comprehensive honeypot site; especially good place to find many different types of honeypot software tools.

http://www.tracking-hackers.com/

Lance Spitzner's website; contains whitepapers that detail many different honeypot and honeynet solutions. Topics range from broad definitions to detailed minutiae, and novice and expert users alike will find a plethora of useful information.

http://www.honeynet.org/

The home site of "The Honeynet Project," it contains links to honeypot challenges and Scan-of-the-Month exercises as well as various security-focused papers written by members of the networking community. The options on extracting data from Snort log files was also found on this site, in "Rain Forest Puppy's" answer submission for Scan-of-the-month 14.

http://www.honeynet.org/reverse/results/project/020601-Analysis-IP-Proto11-Backdoor.pdf

Useful NVP data. Talks about the different commands and where to find info in the packets for each.

Appendix A: Review of how to set up and run imapd exploit

This part is directly taken from the Buffer Overflow lab and it goes over how run the imapd exploit.

Setting up the exploit on the Red Hat 7.2 machine:

Instructions:

Step 1: Installing the 7.2 Operating System

1. Install RedHat Linux (version 7.2 server configuration)
2. Disable the firewall
3. Make sure the machine has a static IP address assigned to it.

Step 2: Starting the imap server and running the exploit

1. Copy the imapd_exploit.tgz file from the NAS (under the Forensics folder) onto the machine
2. Upzip the file

```
tar xvf imapd_exploit.tgz
```

Step 3: Now, you will have a folder called *imapd_exploit*. Go to this folder

```
cd imapd_exploit
```

Step 4: Run the following commands

4.1 `nmap localhost` or `nmap the local static IP`

This will give you a list of the currently running services on the machine. Note there is no imap2 running.

4.2 `make`

4.3 `make install`

- this command will copy imap into /etc/xinetd
- install imapd RPM
- make netcat (install to /usr/sbin)

4.4 `make restart (to restart xinetd)`

4.5 `nmap localhost`

Make sure imap2 is running on the machine. It should be in the list that is displayed

4.6 `make run`

- run the exploit with a proper offset (sample: 3000, 4000)
- ls (gives list of files)

Note: Now you have successfully executed the exploit and obtained a list of files from the victim machine

4.7 `make uninstall`
4.8 `make clean`

About the offset: The exploit code needs netcat and cat in combination to perform the buffer overflow. The exploit code makes use of a string of 942 NOP instructions and allows the user to input an offset from which to attempt the attack. Normally Offsets range from 3000 to –3000

The command string
(./imapd-ex 0; cat) | nc localhost 143
can also be written as
(./imapd-ex 0; cat) | imapd
to test the exploit on the local imapd (imap deamon)

- netcat
  a.k.a. the swiss army knife of the Internet. netcat has many functions. One of which is the ability to send streams of data to a destination. netcat (/usr/sbin/nc) is included as part of the install for this exploit.

Running imapd exploit from a remote machine (our Red Hat WS 4.0 machine):

- Install the imap server on the victim machine (described above)
- Copy the *imapd-ex* file to the attacking machine (In our case the Red Hat 7.2 machine)
- Execute the following command

```
(./imapd-ex 3000 ; cat) | nc a.b.c.d 143
```

where,
3000 = offset (this could be different. Try with various offsets)
a.b.c.d – static IP of the victim machine (your Linux 7.2 IP)
143 = port number

Appendix B: NVP excerpt

from: http://www.honeynet.org/reverse/results/project/020601-Analysis-IP-Proto11-Backdoor.pdf

Technical Description

The IP Protocol 11 (NVP) Backdoor Tool is a utility that receives commands through a protocol designed to masquerade as NVP (Network Voice Protocol) traffic. It is designed to give a remote attacker the ability to control a machine through non-standard communication channels, and is able to take advantage of permissive firewalls that allow IP Protocol 11 packets to pass through unfiltered.

All communications are encoded with a custom-encoding algorithm. In the event that a communication packet contains commands for direct execution by the infected host, this encoding system prevents an eavesdropper from easily obtaining the commands in plaintext as they travel across intermediary networks. Additionally, the communications protocol is entirely stateless, thereby allowing an attacker to mask his identity by spoofing the source address of his communication.

All data sent and received from this remote control utility is contained within IP packets with the 8-bit protocol field set to `0x0Bh` (11). The SecurityFocus Threat Analyst Team believes that this communication technique was designed to avoid filtering from improperly configured firewalls and evade detection by intrusion detection systems (IDS).

Once executed, the backdoor tool opens a RAW socket to listen for incoming data marked as IP Protocol `0x0Bh` (11) in the IP header. Upon receiving incoming data, the parent process will decode the incoming data and will key on the second byte of the decoded payload in order to determine the requested course-of-action. If appropriate, the parent will then fork() a child process to complete the requested command, allowing the parent process to continue listening for further communications. The parent/child design used by this utility allows an attacker to maintain control of the machine in the event that a child process dies, or stops responding. Additionally, several of the DoS functions that are used by the child processes within this program will continue iterating indefinitely until the process is killed.

In the event that the attacker requires an actual response, he is able to instruct the utility to respond to a specific IP address by passing it in an encoded packet with the `0x02` / Initialization command selected in the command byte of the decoded payload. In order to obfuscate the attacker's location when responding to this address, the utility will, at the option of the attacker, respond to multiple random hosts in addition to the host specified by the attacker.

As analyzed, the server recognizes twelve (12) distinct commands passed in an 8-bit field in the second byte of the decoded payload. Details regarding these commands are detailed below:

`0x01h` Query server for status information This command instructs the server to generate a response indicating the child process PID, if any, as well as the command number that the child process is currently executing. It may also report the list of random IP addresses that are being used for responses, as well as additional information about the infected host. Indications from the initial analysis show that the destination address is either randomly chosen, or used from the list generated by the `0x02h` command.

`0x02h` Initialization and attacker IP adjustment This command will perform several actions. First, the infected host's IP address (determined by the destination address in the IP header) is stored in memory for later reference. Additionally, an IP address is specified within the decoded payload (as bytes 4 to 7, inclusive) that the server will use as the destination address

for all subsequent responses. There is a special option within this command that instructs the server to create an array of ten IP addresses, of which all but one randomly chosen
entry (containing the IP address specified within decoded payload) will include randomly generated IP addresses. The random number generation used by the IP Protocol 11 backdoor does not appear to be accomplished via standard GNU calls to srandom() or random(). In GNU C, random() uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return random numbers. Similarly, srandom() sets its argument as the seed for a new sequence of pseudo-random integers to be returned by random(). The random number generator used in the backdoor, on the other hand, appears to be based on a seeded engine that uses a dynamic look-up table, bit shifting, and other basic mathematic operations. Although this random number generation routine is similar to srandom() and random(), the SecurityFocus Threat Analyst Team was unable to reproduce a similar random number generation algorithm using GNU implementations of srandom() and random(). This feature allows the attacker to obfuscate the IP address that he is using to listen for responses by forcing the server to send out multiple responses to random IP addresses, only one of which will actually be destined for the location specified by the attacker.

`0x03h` Execute specified commands via `/bin/csh`, and respond with output This command instructs the server to fork() a child process and execute the supplied commands (encoded within the packet) via `/bin/csh`. Output from this command is redirected to a temporary file, "`/tmp/.hj237349`", and after execution has completed, this file is opened and the contents of it are sent as a response. Indications from the initial analysis show that the destination address is either randomly chosen, or used from the list generated by the `0x02h` command. The file containing the output is then removed from the system via an unlink() call.

`0x04h` UDP Flooder Using DNS Reflection Attacks This command instructs the server to fork() a child process, and initial analysis suggests that the child will attempt to utilize an internal list of DNS servers as intermediary hosts in a DNS Reflection attack against a user-specified target. It appears that a small delay is initiated after each packet is sent out. Nearly all fields in the IP header and UDP header are randomly generated, and filtering or identifying the packets responsible for this attack based on header information alone is very difficult; typically, the only consistent data within the network layer protocol header is the UDP protocol identifier, `0x11h` (17). The transport layer protocol header is similarly varied, with only the source port 53 (DNS) remaining constant.

`0x05h` UDP or ICMP Attack This command instructs the server to fork() a child process, and initial analysis suggests that the child will flood specified IP addresses with either UDP or ICMP flood attacks. The attacker specifies the type of attack in the command packet, either UDP or ICMP. The ICMP packets generated by this attack consist of type 8, code 0, or ECHO_REQUEST packets, and the UDP datagrams contain a destination port specified by the attacker. Packets generated by this command contain spoofed source addresses, and initial analysis suggests that they are a combination of user-specified and randomly generated addresses.

`0x06h` Open password-protected portshell on TCP port 23281 This command instructs the server to fork() a child process and listen for a TCP connection on port 23281. Upon connecting, it issues a single call to recv(), and checks for the ASCII string "`SeNiF`" followed by `0x10h` or `0x13h` before spawning an instance of `/bin/sh` and binding the standard file descriptors to the open socket. It should be noted that due to the fact that there is only one call to recv(), the entire password must be present in the infected host's receive buffer when the recv() call

stops blocking. Thus, under normal circumstances, this password cannot simply be sent interactively with a keystroke-by-keystroke protocol, such as the default communications method in most telnet clients.

0x07h Execute specified commands via `/bin/csh` This command instructs the server to fork() a child process and execute the supplied commands (encoded within the packet) via `/bin/csh`. Output from this command is discarded.

0x08h Signal child process, if any, with `SIG_KILL` This command instructs the server to signal the child process, if any, with `SIG_KILL`, thus causing it to terminate. The child process PID is typically stored in a global variable when forked, allowing this command to terminate a hung process. Additionally, most commands check for an active child process before following through with forking, and will abort such an action if a child process is already active.

Appendix C: Set up and use SnortALog to analyze Snort logs.

(http://jeremy.chartier.free.fr/snortalog/)

Note: This appendix is not required for the lab; however, this tool is used by professionals on a daily basis. Familiarity with SnortALog may prove helpful in future jobs.

One of the biggest problems with using a product like Snort is that on any production level network you will receive a huge amount of data in your Snort logs. In some cases, Snort logs can be megabytes in size and thousands of lines of events. Even with a tool like Ethereal, it can be very time consuming to go through the logs and track malicious activity. One possible solution is to use a product called a "log analyzer tool" which runs scripts on a Snort log and produces summary information and graphs that can help with the anlaysis. One such tool and the one we will use in this lab is called SnortALog, and is downloadable from: http://jeremy.chartier.free.fr/snortalog/#download.

SnortALog is a PERL script that takes in a unified Snort log file and generates a graphical report that shows summary information in an HTML format. The report illustrates such items as: distribution of attacks by protocol, start times, end times, distribution of attacks by hour and by day, percentage of attacks by port number, host sources and domain resolution, and attack summaries based on method (as well as much more info).

To set up the basic configuration of SnortALog, download the file – "**snortalog_v2.2.0.tgz**" from the website:
http://jeremy.chartier.free.fr/snortalog/#download

Save the file in your /home/tools directory on your RedHat 4.0 Workstation. The RedHat 4.0 Workstation must be used for this because the program requires Perl 5.8.x and the RedHat 7.3 Servers are running Perl 5.6.x.

On RedHat 4.0 WS, type the command: **# tar xvfz snortalog_v2.2.0.tgz**

This will create a folder called "snortalog_v2.2". The base installation consists entirely of extracting and unzipping the tarball to our workstation, the program runs directly from a PERL script.

The next step is to download and install some extra software packages that will be used in the generation of the HTML report and the creation of some graphics like pie charts and line charts that show Snort attack information.

Download the following files, all of which are obtainable from the following website:
http://jeremy.chartier.free.fr/snortalog/config.html#dependencies
Except for the first one which is downloadable from the website referenced after the file name.

gd-2.0.11.tar.gz (PNG and JPG format) http://www.boutell.com/gd/

gd-1.19.tar.gz (Generates GIF format files)

GDTextUtil-0.85.tar.gz

GDGraph-1.39.tar.gz

htmldoc-1.8.23-source.tar.gz

HTML-HTMLDoc-0.07.tar.gz

Install all of the above files by following these procedures. It is important to install these in the order listed and by using the commands exactly as they are listed.

GD-1.19.tar.gz

```
# tar xzvf GD-1.19.tar.gz
# cd GD-1.19
#
# perl Makefile.PL
Checking if your kit is complete...
Looks good
MakeMaker (v6.03)
Writing Makefile for libgd
Writing Makefile for GD
#
# make
# make install
```

GDTextUtil -0.85

```
# tar xzvf GDTextUtil-0.85.tar.gz
# cd GDTextUtil-0.85
#
# perl Makefile.PL
Checking if your kit is complete...
Looks good
Writing Makefile for GD::Text
#
# make
# make install
```

GDGraph-1.39

```
# tar xzvf GDGraph-1.39.tar.gz
# cd GDGraph-1.39
#
# perl Makefile.PL
Checking if your kit is complete...
Looks good
Writing Makefile for GD::Graph

The automatic tests for GDGraph are not really a solid workout of
the library. The best way to test the package is to run the
examples before installing it. You can run the examples in the
samples directory with `make samples` or by going into that
directory, and just running `make`.
If that fails, please read samples/Makefile.
#
# make
# make install
```

Gd-2.0.11

```
# tar xzvf gd-2.0.11.tar.gz
# cd gd-2.0.11
#
# ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
...
checking for jpeg_set_defaults in -ljpeg... yes
checking for XpmReadFileToXpmImage in -lXpm... no

** Configuration summary for gd 2.0.11:

  Support for PNG library:            yes
  Support for JPEG library:           yes
  Support for Freetype 2.x library:   yes
  Support for Xpm library:            yes

configure: creating ./config.status
config.status: creating Makefile
config.status: creating config/Makefile
config.status: creating config/gdlib-config
config.status: creating test/Makefile
config.status: creating config.h
config.status: executing depfiles commands
#
```

```
#make
#make install
```

HTMLDoc -1.8.23

```
# tar xzvf htmldoc-1.8.23-source.tar.gz
# cd htmldoc-1.8.23
#
# ./configure
# make
# make install
```

HTML-HTMLDoc -0.07

```
# tar xzvf HTML-HTMLDoc-0.07.tar.gz
# cd HTML-HTMLDoc-0.07
#
# perl Makefile.PL
Checking if your kit is complete...
Looks good
Writing Makefile for HTML::HTMLDoc
#
# make
# make install
```

That finishes up all of the installation procedures. Now SnortAlog is installed with all of its dependencies and it is ready to use. But first, it is necessary analyze a log file. To do this, create a new rules file on the RedHat7.2 Virtual Machine that will scan for TCP traffic.

On the RedHat7.2 Virtual Machine, go to the rules directory located in the root Snort directory (probably at /root/snort-1.9.1/rules/) and create a new file called **test.rules**. Open **test.rules** in a text editor and enter the following rule:

```
alert tcp any any -> any any (msg:"TCP Traffic";)
```

Now run Snort with this new rule file and collect some log data. The command for this is:

```
# ./snort -A full -l ./log -c ../rules/test.rules
```

With Snort running on the RedHat7.2 Virtual Machine, go to the Red Hat 4.0 Workstation and generate some TCP traffic. Do this by opening a telnet session on the RedHat7.2 Virtual Machine and entering some commands. After running some simple telnet commands, go back to the Snort application and kill it with CTL-C. There should now be a log file present in the log directory.

With this log file, it is now possible to run SnortAlog and generate a report.

FTP the log file back to the Red Hat 4.0 Workstation and copy it into the snortalog_v2.2 directory. Then, run the following command:

```
# cat snort.log | ./snortalog.pl -r -n 30 –report –g gif –h
test.html
```

This command will cat the log file and pass the contents to the snortalog.pl PERL script and dump the contents of the report to a file called test.html which will be located in the snortalog-v2.2 directory. This HTML file will include GIF format graphics as well as many other pieces of information.

Open the HTML file in a web browser and examine the contents

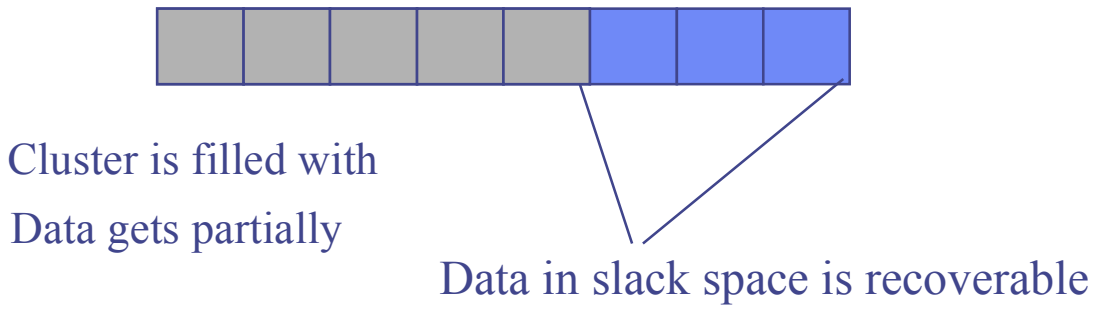
Appendix D: Digital evidence: Today's fingerprints

Lab 7 lightly covers forensics tools and only mentions that it is possible to recover deleted files from the hard drive. The lab primarily focused in fingerprinting the system and then detecting what files were modified, deleted, if any. This lab addition brings a more in-depth view of forensics and explains why and how data can be recovered even after deleting it. Furthermore, we will evaluate the forensics software for windows: Back2Life.

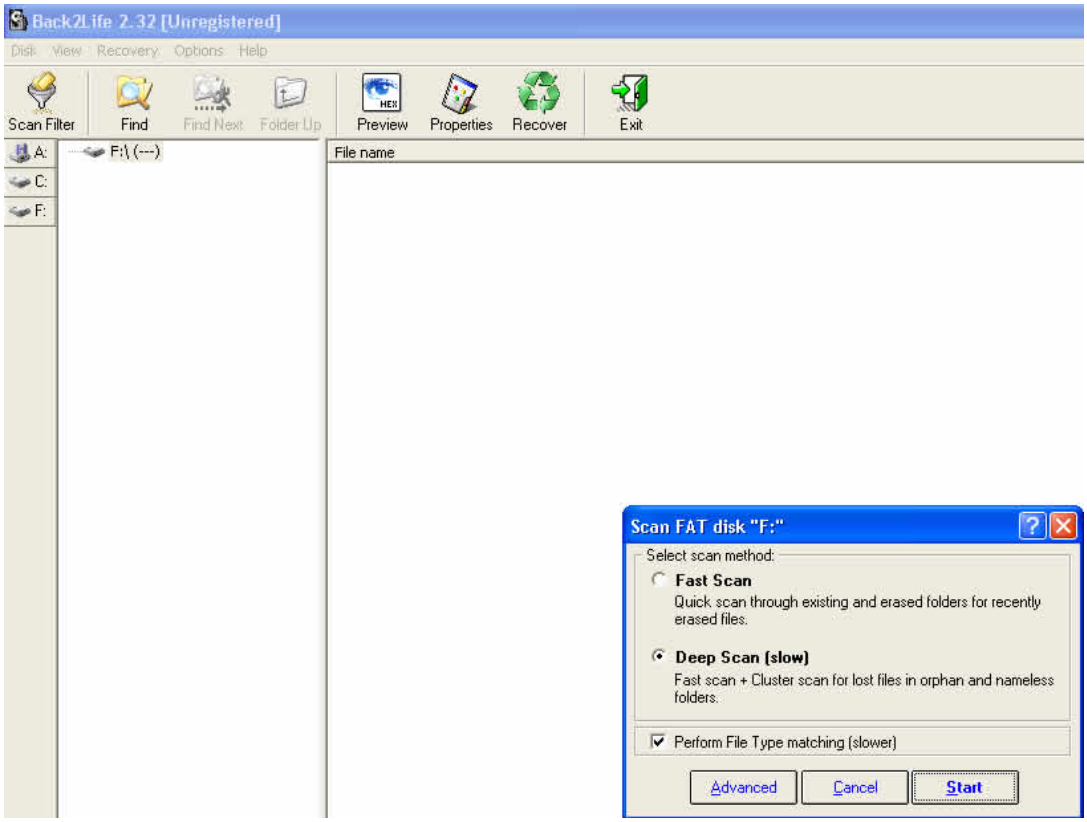
Forensics tools allow a user to gather, retrieve and present data that was previously deleted. It is a science in the sense that forensic imaging must follow certain recognized standards and protocols. It is an art in the sense that no two forensic examinations are the same although they should arrive at the same conclusion. Basically, it can help you recover "deleted" files on your computer and instead of just knowing what files were modified by a hacker on your system, you can find what tools he used.

Deleting a file doesn't really erase it. Even after emptying the Recycle Bin, that's not the end of it. Files "deleted" by the operating system aren't erased from the disk. All that happens is that the pointers to that file are removed from the file system's table and the space where it is stored on disk is marked as reusable. The 1s and 0s that make up the data itself are still sitting there until new data is written to that location. In simple terms, the pointers to the data are gone, but the data remains. And even when new data is written to it, because the drive heads don't always line up exactly the same every time, fragments of the old data may still be discoverable.

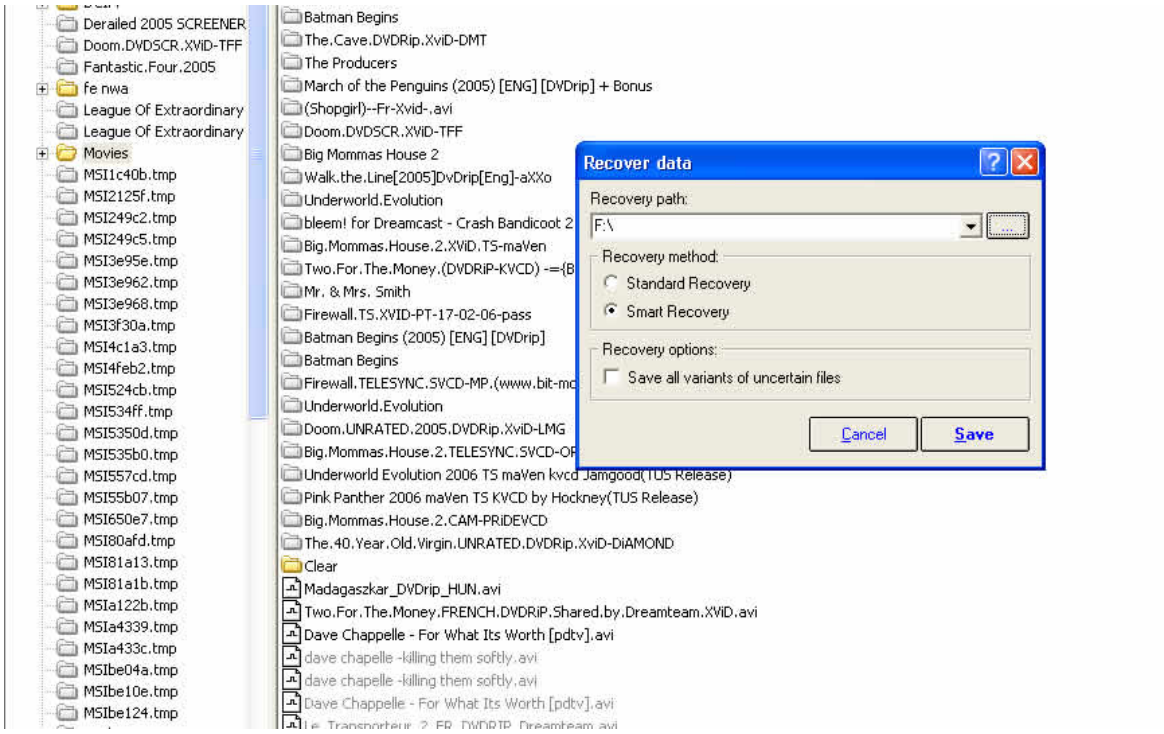
In Windows, data is stored on the hard drive in "clusters". Since these clusters are of fixed-length, even if data being stored doesn't completely fill the cluster, the entire cluster is used for the file. The remaining unused space is called slack space. Theoretically, this data could stay on the hard drive forever!



The forensics tool Back2Life was used to demonstrate a simple “undelete” process. Back2Life is user-friendly and intuitive to use. Looking and acting similar to Windows Explorer, it provides recovery access to erased folders and lost files, which can be treated just like normal ones.



It is important to note that one should not install Back2Life on the drive where we are trying to undelete the files. At the program startup, on the left, we choose the drive that contains the files that we want to recover by clicking on it. We then click on the button “Scan Filter”. This will open a new window that will give us the choice between a fast scan and a deep scan. Once the mode has been chosen, we just click “Start” and the software will scan the hard drive to find previously deleted files.



Once the scan is over, we can see a list of previously deleted files. In order to recover the files, we only need to choose the files and the recovery path and click the "Recover" button at the top. As expected, the file will be restored! It is not recommended to recover the file on the same hard drive that it is being recovered from.

In order to safely delete a file:

- Delete file using "shift + delete" to bypass the recycle bin
- Delete using file shredding utility (only deletes file, not associated files)
- Defrag drive (formatting the drive will NOT delete the data)
- Use disk wiping utility (Simple overwrite, DOD – 7, Guttman)
- Wipe Swap File (should use fixed size for swap file)

If you will be giving away or selling your computer and you're worried about sensitive data remaining on the drive, just formatting it is not enough. Use an overwriting program to overwrite the disk **multiple times**.

Sources:

http://www.windowsecurity.com/articles/Sensitive-Data-Around.html

http://www.gsforensics.com/Intro.pps

http://www.cs.jhu.edu/~coulls/ch08.ppt

Appendix E: Basic Analysis of Windows Shellcode Through the Usage of the Malcode Analyst Pack

Background:

The primary motivating factor for assisting us in shellcode analysis is that honestly what does the following sequence mean to you "\xaf\x42\xe6\x5d\x36\xfc\xb3\x3c\x38\xe3\xf3\x3c\x0f\xc0\x7f\xde"? This is only a small fragment of a basic shellcode used to create a bind back shell.. Even if you were to disassemble the hex it is very difficult to follow, unless you are very good with x86 assembly. As incomprehensible as that line was that is what you will find in a log for an IDS snort, or honeypot. Anyway that sequence of bytes may actually be XOR encoded to prevent IDS signature detection and may not actually represent the actual shellcode payload.

However you don't want to just let a shellcode run either without control or monitoring, you want to be able to see what the code does as well as prevent it from damaging your system or other systems connected to it, even if it is just a virtual machine.

Thus bringing us to the sandbox; the idea behind the sandbox is you provide a strictly controlled environment from which you can execute shellcode. It solves a multitude of problems. It solves the problem of not having the decoded shellcode as it allows the decoding routine to run on the shellcode, as well as solving the control problem by limiting the system calls that the shellcode can make. Sandboxes should be used in combination with disassemblers in the analysis of shellcodes to provide an easier way of determining what a shellcode does.

Objective:

The objective of the lab is to show how to use a tool such as sclog to perform shellcode analysis in a safe, controlled environment. We will do this by running sclog on some shellcode examples provided to us by Idefense with the installation of sclog as part of the Malcode Analyst Pack, as well as if desired on some shell code created by Metasploit.

Tools that are useful for shellcode analysis (not all required for lab):

Hexeditor and Disassembler

- An easy to use Hexeditor and Disassembler is Techno Logismiki Hackman suite found http://www.technologismiki.com/en/index-h.html, however any hex editor and disassembler will suffice.

- The purpose of the hex editor is to reformat a hex display both as ascii values and just to get a feel of if there is plain text values in the shellcode
- The disassembler is used to interpret the machine code (in hex format), and translate it into the respective instruction in assembly. A word of warning though about using a disassembler, often code in shell code is XOR encoded and may change when actually run.

Supporting Software

- Both of these tools provided by Techno Logismiki requires the Visual Basic Runtime environment to be installed

"Sandbox"

- The sandbox provides a limited environment for the shellcode to execute in, in addition to giving us a view of what the shellcode is actually doing
- For our purposes we will be using the program called sclog created by Idefense which is available in the Malcode Analyst Pack found http://labs.idefense.com/labs-software.php?show=8

Hex to Executable Conversion

- Often the shellcode found by our honeypots is presented in hexadecimal format which is not a valid input to sclog we need a tool to convert the hex numbers (which are human but not machine readable) into an executable format
- Use Hex2Exe also found in Malcode Analyst Pack

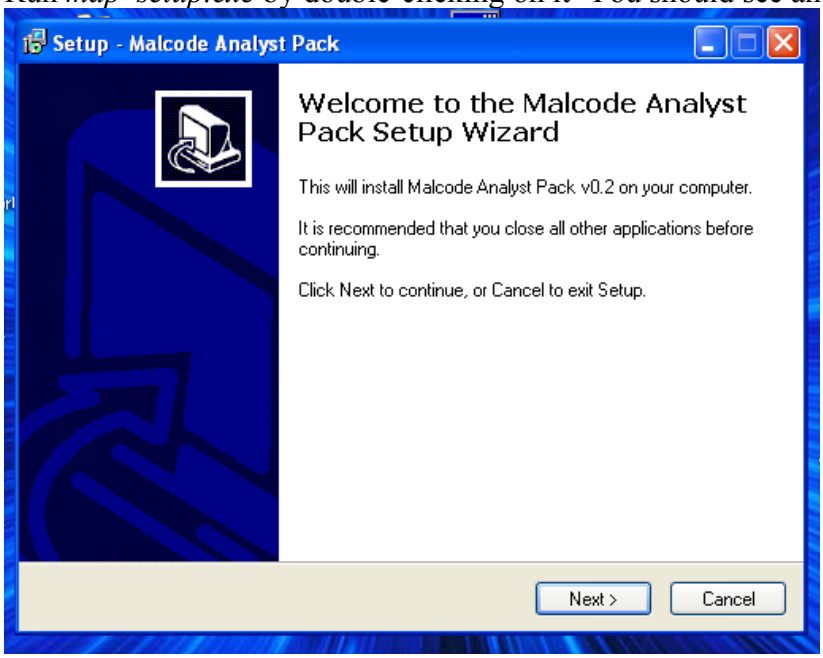
Supporting Software

Hex2Exe is actually and php file and thusly requires a php enabled webserver to run. Thus if you do not have a php enabled server download Xampplite to run an easy to configure PHP/Apache server. Xampp can be found at http://www.apachefriends.org/en/xampp-windows.html#646

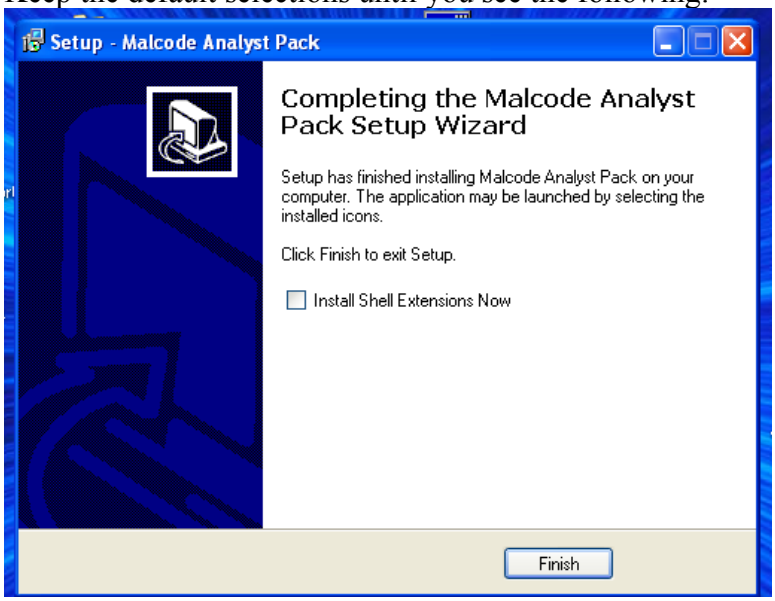
Tools to get:

In order to perform this lab first obtain the Malcode Analyst Pack from iDefense labs at http://labs.idefense.com/labs-software.php?show=8.

1. Save the *map_setup.exe* program to your WindowsXP box.
2. Run *map_setup.exe* by double-clicking on it You should see an image as follows:


Setup - Malcode Analyst Pack
Welcome to the Malcode Analyst Pack Setup Wizard
This will install Malcode Analyst Pack v0.2 on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel

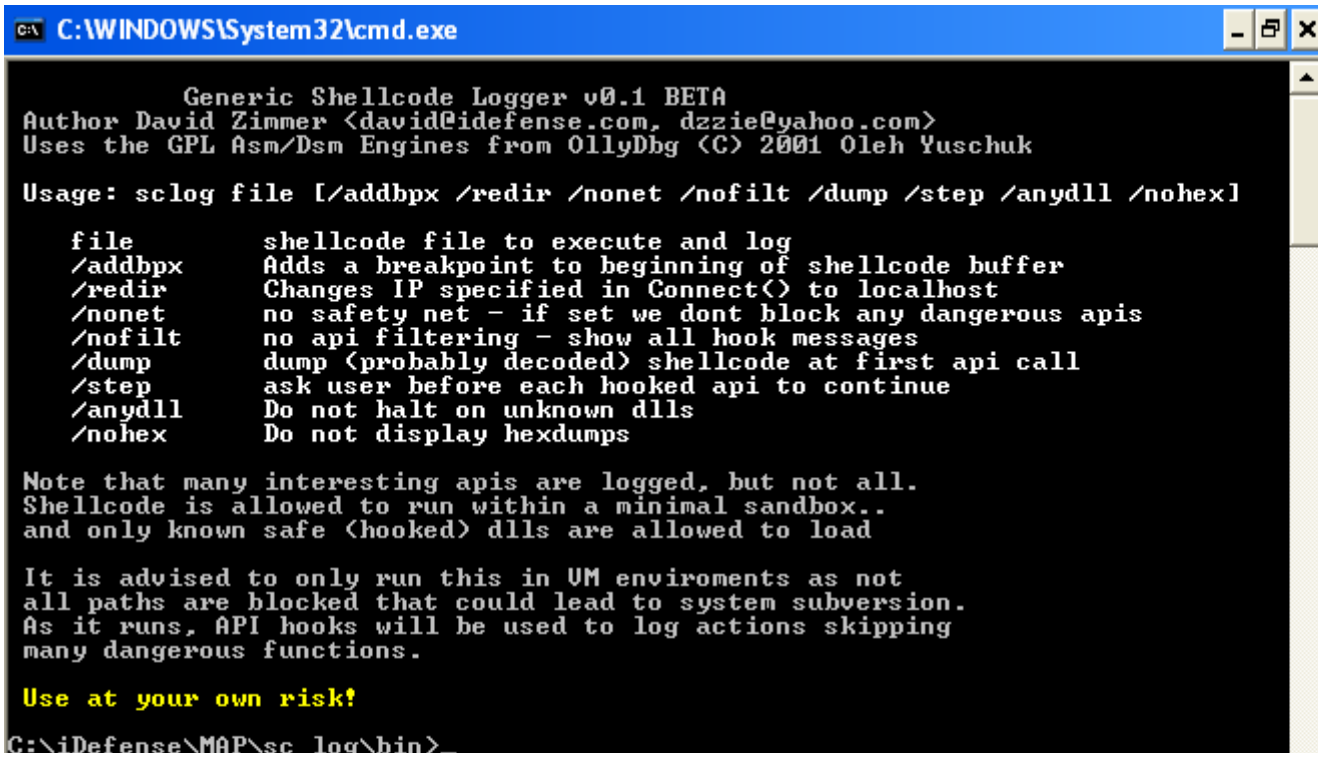
3. Keep the default selections until you see the following:


Setup - Malcode Analyst Pack
Completing the Malcode Analyst Pack Setup Wizard
Setup has finished installing Malcode Analyst Pack on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
Install Shell Extensions Now
Finish

4. Uncheck the "Install Shell Extensions Now". The shell extensions though useful for other applications are not necessary to perform the lab.
5. Press "Finish" to exit the installation.

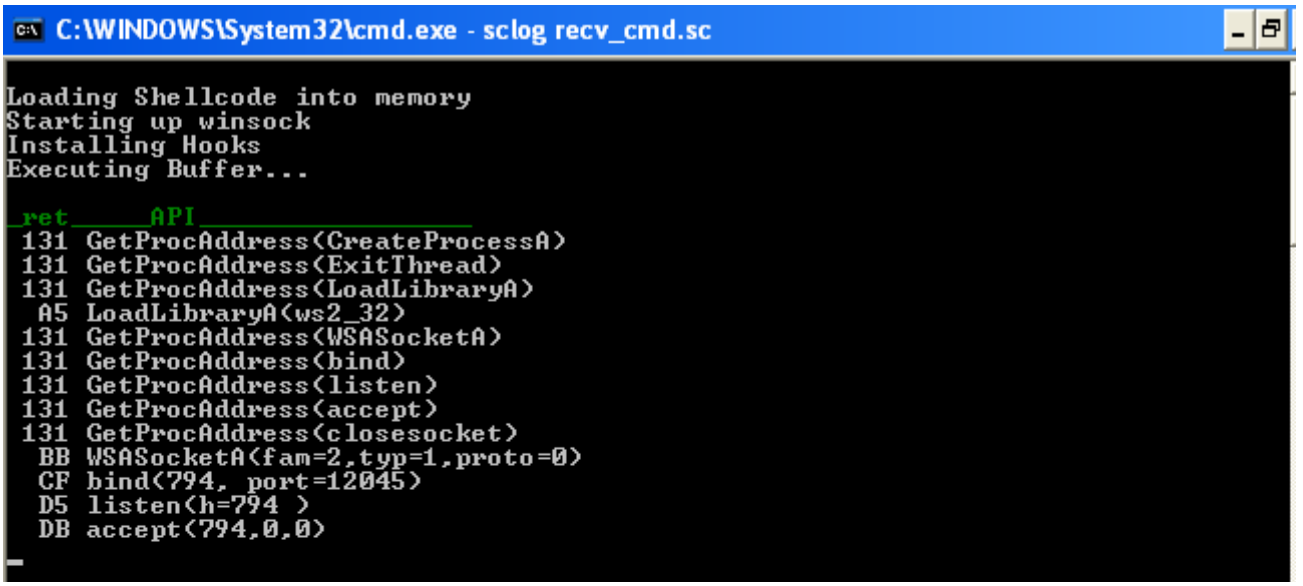
To see an example of the sclog in action:

1. Click “Start” → “Run”
2. Type “cmd” in the box and press okay
3. Navigate the command line to where sclog is installed: the default location is “C:\iDefense\MAP\sc_log\bin”

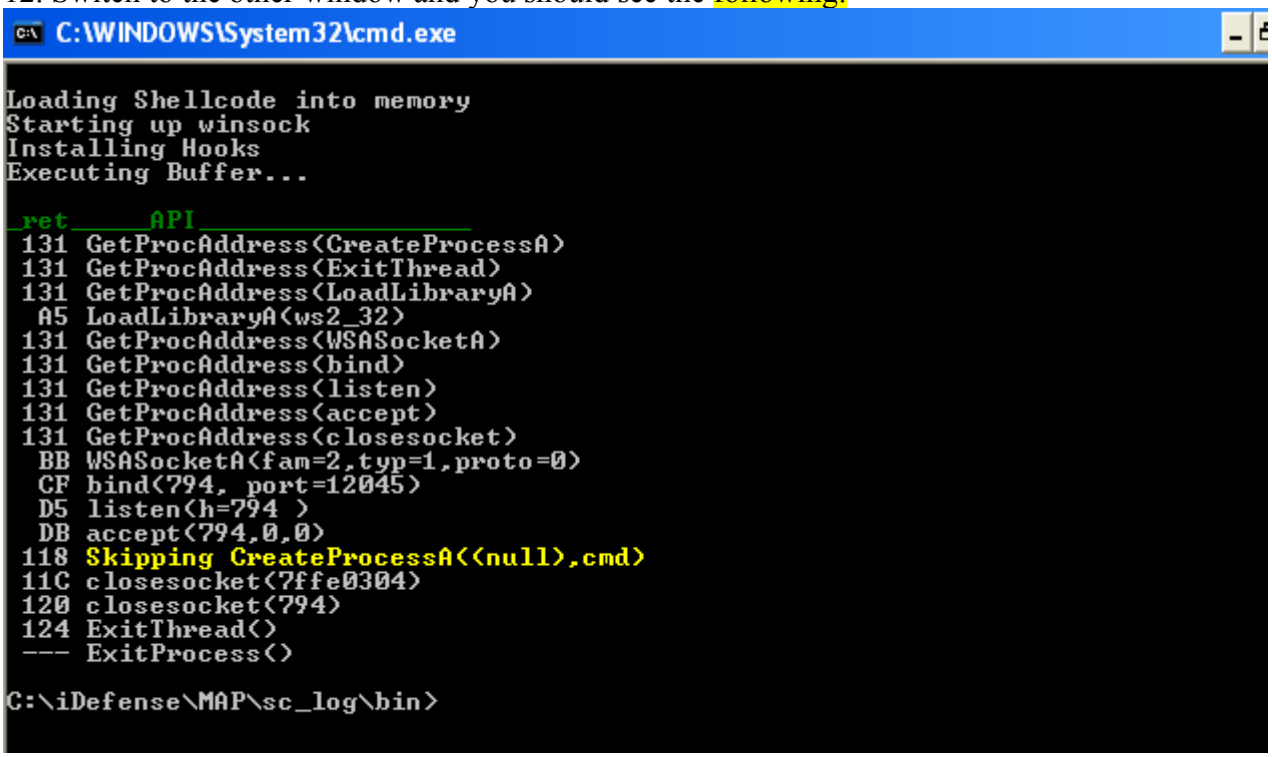

C:\WINDOWS\System32\cmd.exe
Generic Shellcode Logger v0.1 BETA
Author David Zimmer <david@idefense.com, dzzie@yahoo.com>
Uses the GPL Asm/Dsm Engines from OllyDbg (C) 2001 Oleh Yuschuk
Usage: sclog file [/addbpx /redir /nonet /nofilt /dump /step /anydll /nohex]
file shellcode file to execute and log
/addbpx Adds a breakpoint to beginning of shellcode buffer
/redir Changes IP specified in Connect() to localhost
/nonet no safety net - if set we dont block any dangerous apis
/nofilt no api filtering - show all hook messages
/dump dump (probably decoded) shellcode at first api call
/step ask user before each hooked api to continue
/anydll Do not halt on unknown dlls
/nohex Do not display hexdumps
Note that many interesting apis are logged, but not all.
Shellcode is allowed to run within a minimal sandbox..
and only known safe (hooked) dlls are allowed to load
It is advised to only run this in VM enviroments as not
all paths are blocked that could lead to system subversion.
As it runs, API hooks will be used to log actions skipping
many dangerous functions.
Use at your own risk!
C:\iDefense\MAP\sc_log\bin>

4. Once there view the sc_log options by typing sclog.exe

5. After reading over the options available type dir to view the included example shellcodes provided by iDefense
6. We will run sclog on the “recv_cmd.sc” shellcode example.
7. Lets run it with sclog’s default setting by typing “*sclog.exe recv_cmd.sc”* You should now see the following.



8. Can you identify what port it opened?
9. Lets see what happens if we try to connect to the open port.
10. Start another command line window ("Start"→"Run"→ "cmd" → Ok)
11. Lets try telnetting into the open port by typing "telnet localhost <PORT>", replacing "<PORT>" with the port you identified in step 8.
12. Switch to the other window and you should see the following:



13. Note that sclog prevented the shellcode from opening a process.
14. But what happens if we were to allow it to open the process?

How would you run sclog so that it allows the shellcode to create the "cmd" process?

What do you think that the process that it creates does?

But why is this useful at all? Now that we have run sclog we know that the shellcode creates a telnet listener at some port and gives a command line access to any incoming connections on that port. That was a relatively simple shellcode just to test out the functionality of sclog. However, from that we can then examine the dumped shellcode in a disassembler such as Hackman Disassembler and see which lines of code call the functions based on the return address and begin to gain an understanding of how the shellcode operates.

Addition…to the addition…: Real shellcode

It is recommended that this portion of the lab be preformed only if the Shellcode2Exe php script is installed on a web server with PHP support somewhere on the mini-net as since the installation of a PHP supported web server for this one exercise would be waseteful and unnecessary.

In our tests we found we had to modify the line 46 in "shellcode_2_exe.php"from

$arr = array_merge(&$_ENV,&$_GET,&$_POST,&$_COOKIE,&$_SESSION);

To

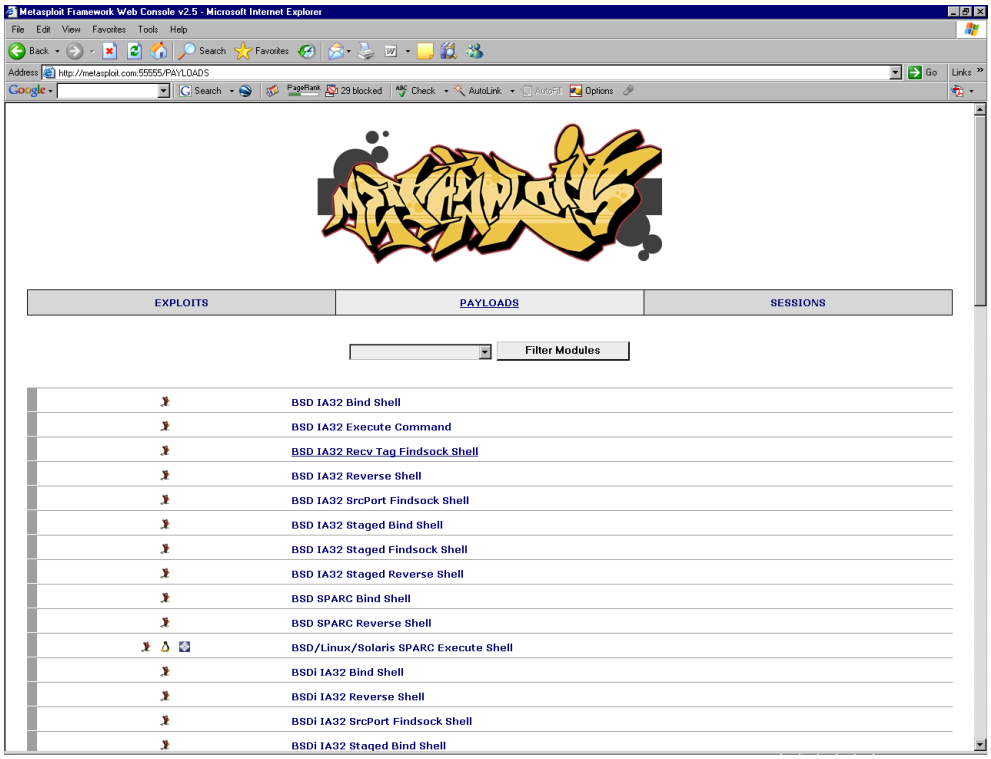
$arr = array_merge(&$_ENV,&$_GET,&$_POST,&$_COOKIE);

In order for the program to function correctly.

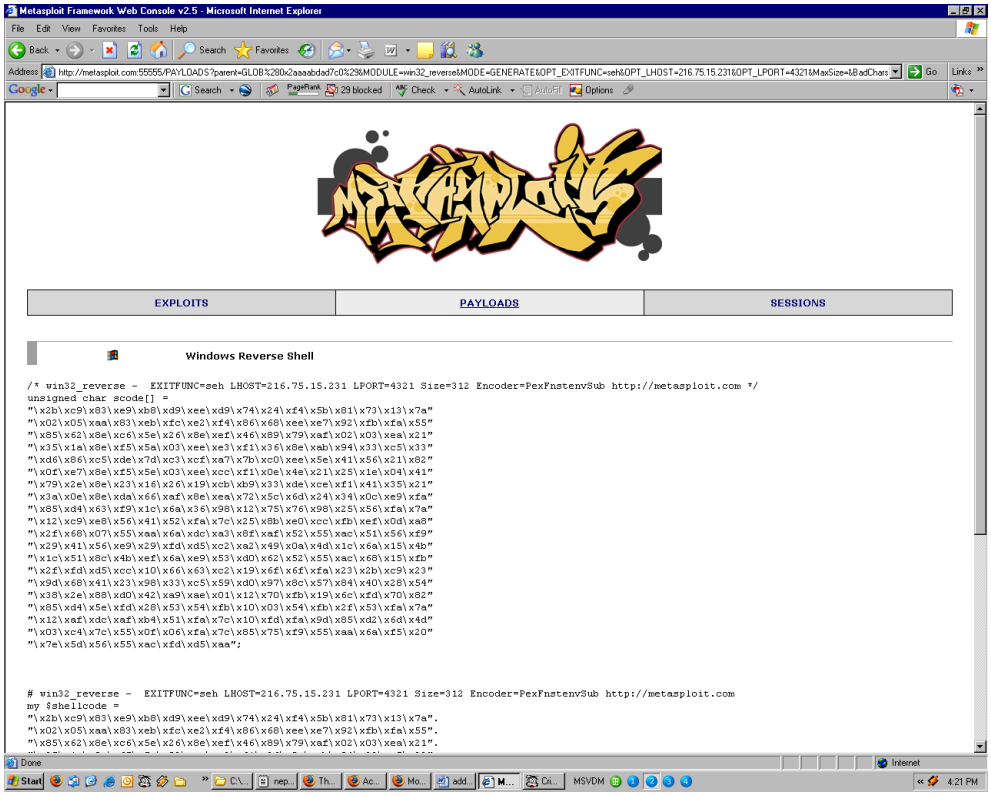
Okay now that was a nice little demo of the basic capabilities of sclog. However, the shellcode that you receive on your IDS systems or on a honeypot will most times will not be a nicely packed executable format for you to pass into the sclog. Real shellcode is found as a series of hexadecimal numbers outputted in the logs of either a honeypot, honeywall, or IDS system and needs to be converted into executable format from human readable format to be practical. These shellcodes are passed into the system most times from bufferoverflow vulnerabilities on web servers and are most times found surrounded by a massive noop sled.

Fortunately the Malcode Analyst Pack contains just a program (or rather script) to do this. This program called Shellcode2Exe is a php script used to convert hex into an executable via a simple webform.

To demonstrate this ability lets find some shellcode to convert. To do this we will use the Metasploit web interface. From a internet connected machine visit: http://metasploit.com:55555/PAYLOADS

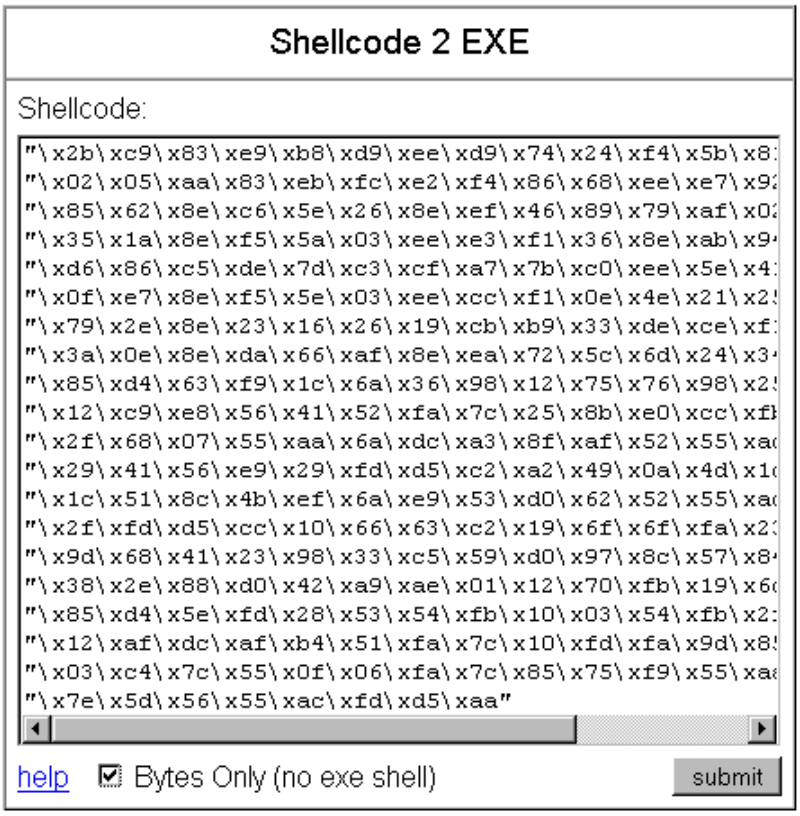

Metasploit Framework Web Console v2.5 - Microsoft Internet Explorer
EXPLOITS
PAYLOADS
SESSIONS
Filter Modules
BSD IA32 Bind Shell
BSD IA32 Execute Command
BSD IA32 Recv Tag Findsock Shell
BSD IA32 Reverse Shell
BSD IA32 SrcPort Findsock Shell
BSD IA32 Staged Bind Shell
BSD IA32 Staged Findsock Shell
BSD IA32 Staged Reverse Shell
BSD SPARC Bind Shell
BSD SPARC Reverse Shell
BSD/Linux/Solaris SPARC Execute Shell
BSDi IA32 Bind Shell
BSDi IA32 Reverse Shell
BSDi IA32 SrcPort Findsock Shell
BSDi IA32 Staged Bind Shell

and choose a windows based payload to generate. Set whatever functions you desire for the payload you choose and generate the payload. You should see something similar to this:



Copy the contents between the “” and browse to wherever the “shellcode_2_exe.php” file is located on the local network.

Paste the shellcode into the text box and select “bytes only”.



Press submit and save the returned “bytes.sc” file to disk and place in the take to your xp machine where sclog is installed to execute the given shellcode.

For more information about Windows shellcodes see:

http://labs.idefense.com/doDownload.php?downloadID=20

http://www.hick.org/code/skape/papers/win32-shellcode.pdf

Appendix F: Web Browser Forensics

Web browser forensics is an article found on internet, which simulate a forensic situation, provided scenario and introduced some forensic process and several web browser forensics tools. The whole article is a mini lab and looks like an great addition to lab 7.

Following is the article which is found on
http://www.securityfocus.com/infocus/1827

With some explanation added, modifications and replaced screen captures (replaced with my own capture)

Tools used:
Pasco
Web historian
Forensic Tool Kit (FTK)

Introduction

Electronic evidence has often shaped the outcome of high-profile civil law suits and criminal investigations ranging from theft of intellectual property and insider trading that violates SEC regulations to proving employee misconduct resulting in termination of employment under unfavorable circumstances. Critical electronic evidence is often found in the suspect's web browsing history in the form of received emails, sites visited and attempted Internet searches. This two-part article presents the techniques and tools commonly used by computer forensics experts to uncover such evidence, through a fictitious investigation that closely mimics real-world scenarios.

While you read this article, you may follow along with the investigation and actually analyze case data. To actively participate in the investigation, you need to download the associated Internet activity data from the SecurityFocus archives [data].

Date is at : http://downloads.securityfocus.com/downloads/JSchmo-InternetActivity.zip

It is a zip file enclosed a person's internet explorer cache

Case notes

At 8.25pm on March 18, 2005, a Senior Associate at a prestigious law firm had just finished a draft of a property-sale contract for his client but was unable to upload the document to the law firm's centralized document storage server hosted by Docustodian, Inc. His attempts to upload the document met with the following error message: "You have reached the storage limit. Please call your system administrator". The Senior Associate did just that, calling Joe

Schmo, the firm's IT administrator. But, Joe's voicemail indicated that he was on vacation from March 7-21, 2005.

This was not an isolated occurrence. An internal review revealed that over 500 GB of MP3s, pirated software, and newly released movies were stored on the system under the profile for Joe Schmo. After finding that a potential intrusion had occurred, the law firm quickly concluded that an investigation of a potential violation of internal policy or an intrusion was beyond their core IT competency and brought in a professional security firm to lead the investigation.

The investigation

During most investigations, an individual's web browsing activity often provides investigative leads. In this investigation, we will begin our analysis by reconstructing the web browsing activity in order to help prove or disprove our suspicions about Joe Schmo, the law firm's system administrator. Our investigation will utilize a combination of commercial and open source tools that you can use to analyze the data provided for this incident. We will walk through their capabilities, how they are used, and what information they will provide us to analyze web browsing activity in this investigation.

Internet Activity File Formats

The predominant two web browsers we encounter during computer related investigations are Microsoft's Internet Explorer (IE) and the Firefox/Mozilla/Netscape family. Each of these browsers saves the web browsing activity (also known as web browsing history) in their own unique formats. We will outline the file formats and the relevant file paths for both IE and Firefox/Mozilla/Netscape's Internet activity files to enhance our investigative leads.

Microsoft's Internet Explorer (IE)

IE is typically installed by default on new Windows-based computers and is used by most private and business computer owners. IE stores the Internet activity for each user under their Windows profile. In Joe's case, since he was using a Microsoft Windows operating system newer than Windows 2000, his IE activity was stored in the following directory:

```
C:\Documents and Settings\jschmo\Local Settings\Temporary Internet Files\Content.IE5\
```

The directory listed above stores the cached pages and images Joe reviewed on his computer. Inside the `Content.IE5` directory there are additional subdirectories, each with a seemingly random name that contains the cached web data Joe had viewed. IE stores this cached information so that Joe does not have to download the same data more than once if he already reviewed the same web page.

We want to point out that there are two additional IE activity directories that may be of interest. The first directory contains the Internet history activity *without* locally cached web content:

```
C:\Documents and Settings\jschmo\Local Settings\History\History.IE5\
```

Under the directory above, there will be additional subdirectories signifying the date ranges where IE had saved the history. The last directory stores the cookie files for IE:

```
C:\Documents and Settings\jschmo\Cookies\
```

An investigator will typically check all three information stores for Internet activity data. If we enter any of the directories presented above, you will find a file named `Index.dat`. The `Index.dat` file contains the Internet activity for each information store. In the cached web pages directory, this file is populated with more information than the others, even though the internal file structures are identical. In order to rebuild a web page a user had visited, the Operating System must find the correct locally cached web page and the corresponding URL the user visited. This relationship is mapped in the `Index.dat` file. This is the same technique we will use when reconstructing Joe's Internet browsing activity. The `Content.IE5` Internet activity directory will be the most useful to us when we reconstruct Joe's activity because we can view the same web pages Joe viewed in the past through his cached versions of these web pages.

The `Index.dat` file is saved in a proprietary binary format that is only officially known to Microsoft. However, the following whitepaper (http://sourceforge.net/projects/odessa) describes some of these internal data structures that may be helpful if you try to reconstruct the file by hand.

Firefox/Mozilla/Netscape Based Web Browsers

Firefox/Mozilla/Netscape and other related browsers also save the Internet activity using a similar method to IE. Mozilla/Netscape/Firefox save the web activity in a file named `history.dat`. One significant difference between a `history.dat` file and an `index.dat` file is that a `history.dat` file is saved in an ASCII format rather than binary. This makes reviewing the file simpler than the corresponding IE file. The second difference with the `history.dat` file is that it does not link web site activity with cached web pages. Therefore, we cannot readily assemble views of web pages Joe visited in the same manner that we can with IE.

Firefox files are located in the following directory:

```
\Documents and Settings\<user name>\Application
Data\Mozilla\Firefox\Profiles\<random text>\history.dat
```

Mozilla/Netscape history files are found in the following directory:

```
\Documents and Settings\<user name>\Application
Data\Mozilla\Profiles\<profile name>\<random text>\history.dat
```

The process of reconstructing web activity manually can be quite tedious. Fortunately, there are several tools, both free and commercial, that streamline this process considerably. The following sections present some of these tools. Please follow along with the web activity data you downloaded in the introduction to this article, and use the tools mentioned in this article to reconstruct the analysis.

Web browsing analysis - open source tools

Pasco

Pasco (the Latin word for "Browse") is a command line tool that runs on Unix or Windows and can reconstruct the internal structures for IE Index.dat files. Pasco accepts an Index.dat file, reconstructs the data, and outputs the information in a delimited text file format. This format

is useful when you need to import the data into a spreadsheet such as Microsoft Excel. Figure 1 shows Pasco in action.

Note on pasco: pasco can be downloaded from
http://sourceforge.net/project/showfiles.php?group_id=78332
To run pasco in windows, no installation from downloaded file is needed.
To run pasco: open cmd in windows, go under the directory bin on pasco,
Type
pasco *history filename* > (an actual arrow character) *target text file*
the generated file can be opened directly in excel with formatted result.

Figure 1. Extracted text file in excel.

Pasco shows that IE saves the following fields from a single web site visit in the Index.dat file:

- The record type - Pasco signifies the activity is either a URL that was browsed or a website that redirected the user's browser to another site.
- The URL - The actual website that the user visited.
- Modified Time - The last moment in time the website was modified.
- Access Time - The moment in time the user browsed the website.
- Filename - The local file name that contains a copy of the URL listed.
- Directory - The local directory you can find the "Filename" above.
- HTTP Headers - The HTTP headers the user received when he browsed the URL.

For each row listed in the spreadsheet, you can retrieve the file listed in "Filename" in Joe's local directory named "Directory" to recreate what Joe saw on the web at the time listed in "Access Time."

Although Pasco works well with IE Internet activity files, it does not reconstruct web activity from other web browsers such as Firefox/Mozilla/Netscape. The output of Pasco as used for this article can be downloaded from the SecurityFocus archives [report].

http://downloads.securityfocus.com/downloads/JSchmo-InternetActivity-Reports.zip

Web Historian

Web Historian can be downloaded at http://www.mandiant.com/webhistorian.htm or from download.com

Red Cliff's freeware tool, Web Historian, has the ability to crawl a directory structure and identify Internet activity files for all of the following web browsers:

- Internet Explorer
- Mozilla
- Firefox
- Netscape

- Safari (Apple OS X)
- Opera

What this means is that the investigator no longer has to memorize the paths for Internet activity files for each web browser. Web Historian also has the ability to output the reconstructed data into the following formats:

- Native Excel Spreadsheet
- HTML
- Delimited Text File

A screenshot of Web Historian in use is shown below in Figure 2.

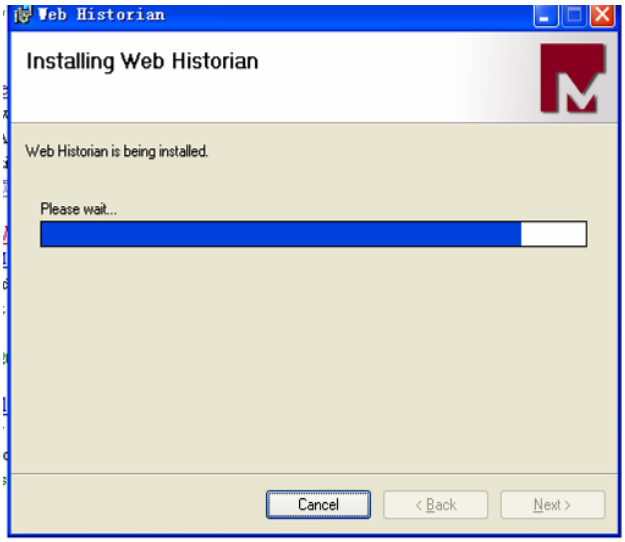


Figure 2. Web Historian in action.

Analysis of the web history

Now that we have the output for Joe's IE Internet activity, we can begin reviewing the websites he visited. During this analysis, we will only present the activity that is relevant to the investigation since there are numerous instances of irrelevant web browsing events that can slow down an investigator. The output from Web Historian is shown below in Figure 3.

Figure 3. Web Historian output.

In the above output we see that Joe visited Hotmail.com. Web Historian shows that the visit to Hotmail created the file named `8R9KCL4N\HoTMaiL[1].htm` in the cache directory. If we open this cached file, we will see the following web page in Figure 4.

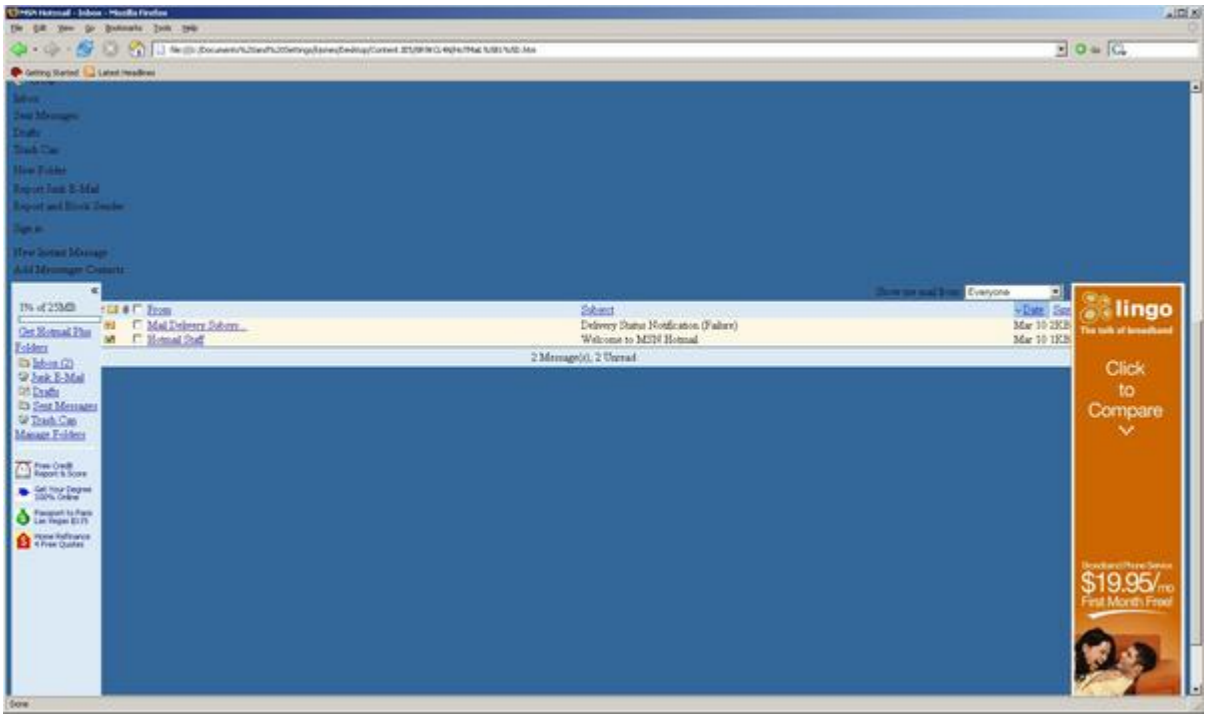


Figure 4. Web Historian reconstruction of Hotmail activity.

At the top of the web page shown in Figure 4, we see that Joe's Hotmail account is `JoeSchmo1980@hotmail.com`. We also see that he does not have any interesting email in this account at the time he checked it.

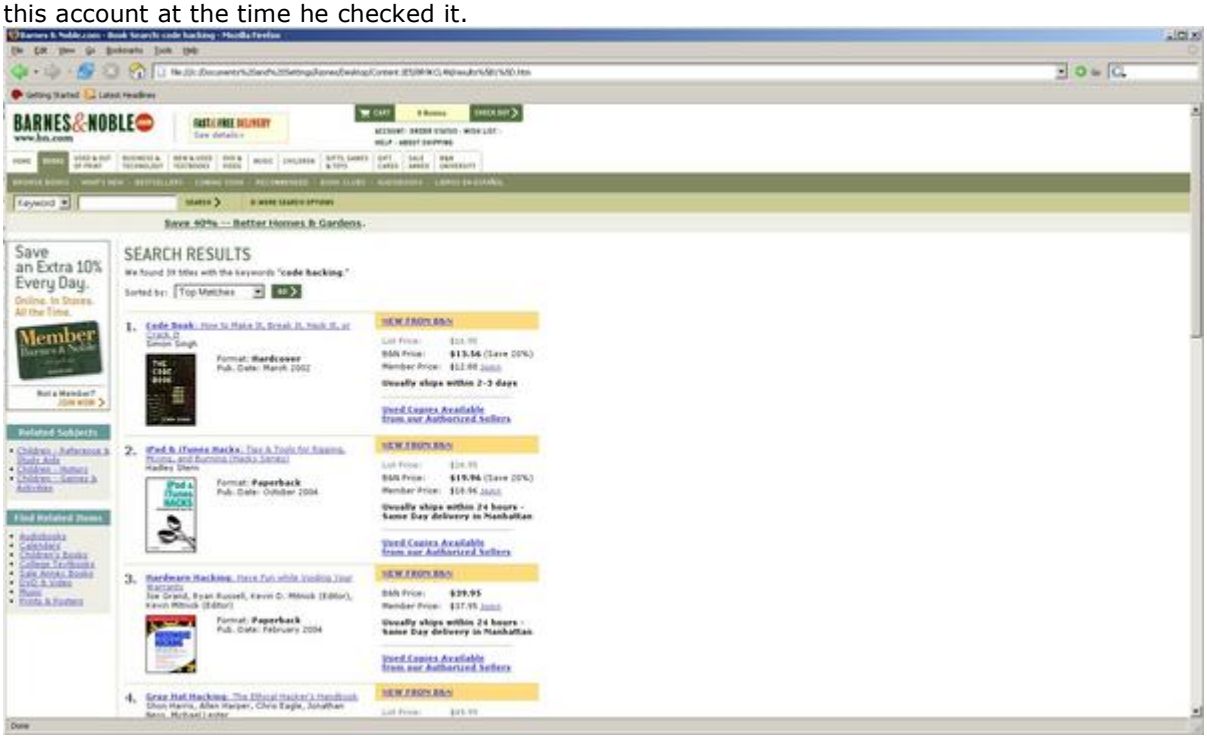


Figure 5. Joe visiting Barnes and Noble.

We see above in Figure 5 that Joe visited Barnes and Noble. It appears as though he is interested in books related to hacking and cracking. There are also other instances of Joe searching for similar material at hacking related websites. In Figure 6 and Figure 7 you will see Joe accessing sites known to have hacking related material. You will also see that Joe is searching for cracks specific to Docustodian, the application that was overloaded with unauthorized material.

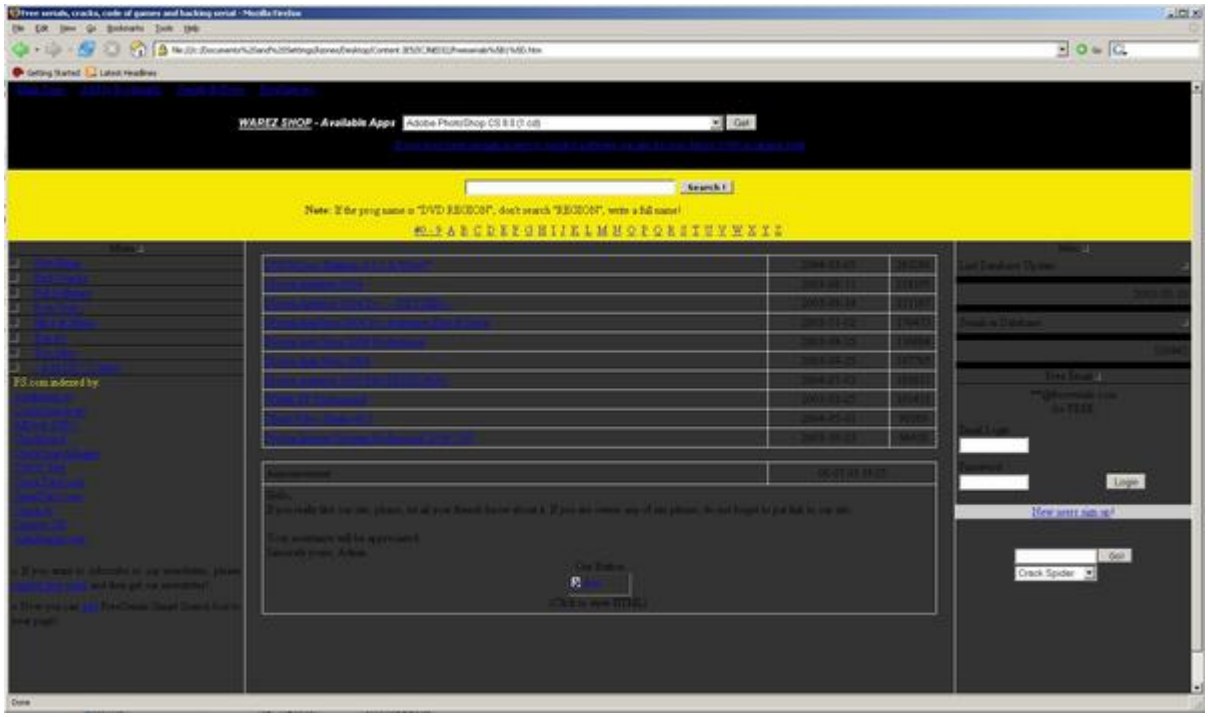


Figure 6. Joe searches for serial numbers.

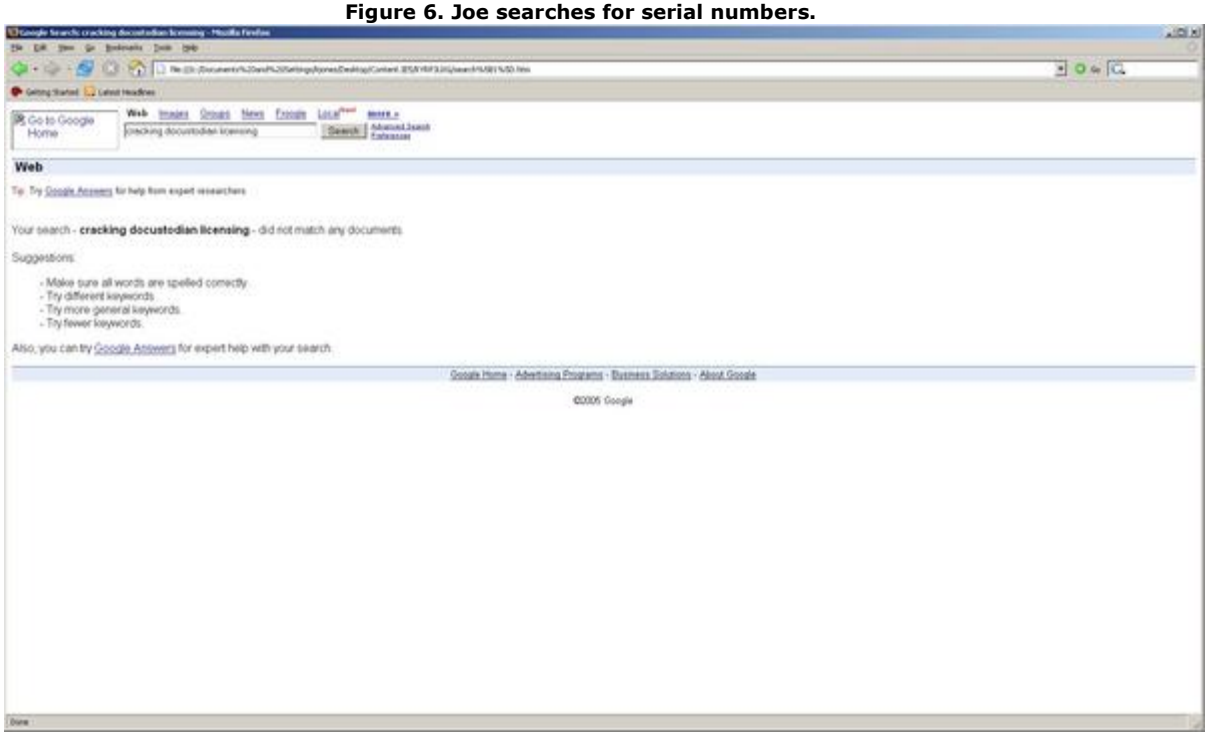

Web
Your search - cracking docustodian licensing - did not match any documents.
Suggestions:
- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.
Also, you can try Google Answers for expert help with your search.
Google Home - Advertising Programs - Business Solutions - About Google
©2005 Google

Figure 7. Joe's Google searches.

As you have seen in the last section, we were able to show that Joe, or someone using Joe's account, was interested in information that would allow him to crack the licensing for Docustodian. However, the time that most of the websites were visited was approximately at 5:50:58 PM on March 10, 2005. It's important to remember that Joe was on vacation from March 7, 2005 through March 21, 2005. It would be highly unlikely that Joe visited these websites from a sunny beach in Florida. We would have to look harder at Joe's computer to see how these websites were accessed.

You can download the output of Web Historian, as previously mentioned in the article.

http://downloads.securityfocus.com/downloads/JSchmo-InternetActivity-Reports.zip

Internet activity analysis - commercial tools

There are several commercial tools that will examine web related activity similar to the freely available tools we presented above. Although we already examined interesting activity in the last section, we will present some of the differences and other interesting web sites Joe's account visited with commercially available tools in this section.

IE History

NOTE: by the time I check, IE history's software company
http://www.phillipsponder.com/histviewer.htm
stated that this program is only available to Law Enforcement and Corporate Security and an written order form is required to get the software. So this section was skipped.

IE History was one of the first commercial tools developed for web activity reconstruction. IE History is a Windows application that opens several types of web browser history files including IE and Firefox/Netscape/Mozilla. IE History is a lightweight tool that can easily export the web browsing history to spreadsheets and text delimited files.

Within IE History, you can open an `Index.dat` file similar to the other tools, as shown below in Figure 8.

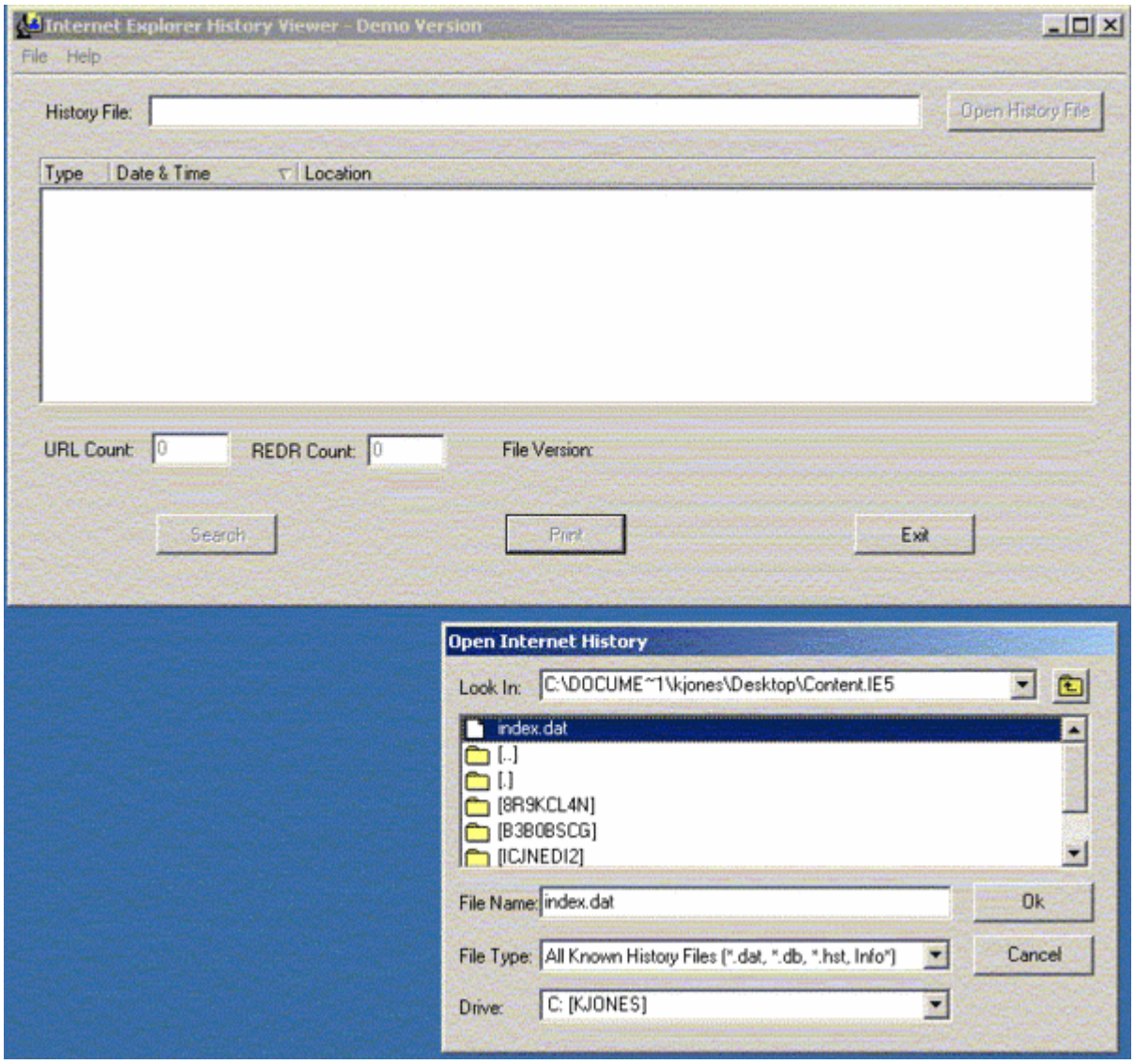


Figure 8. Using IE history.

Once IE History parses the information in the `Index.dat` file, it offers some functionality that simplifies your review. For example, in several instances of activity shown in Figure 9, you can right-click and select "Go to URL" to quickly open a web browser and visit the website Joe visited. Notice, however, that IE History does not link the web activity to the relevant cached files. This means that when you review the URL through the right click function, you are actually viewing a live copy of the website. This means that you may be seeing a different view than Joe when he visited the website in the past.

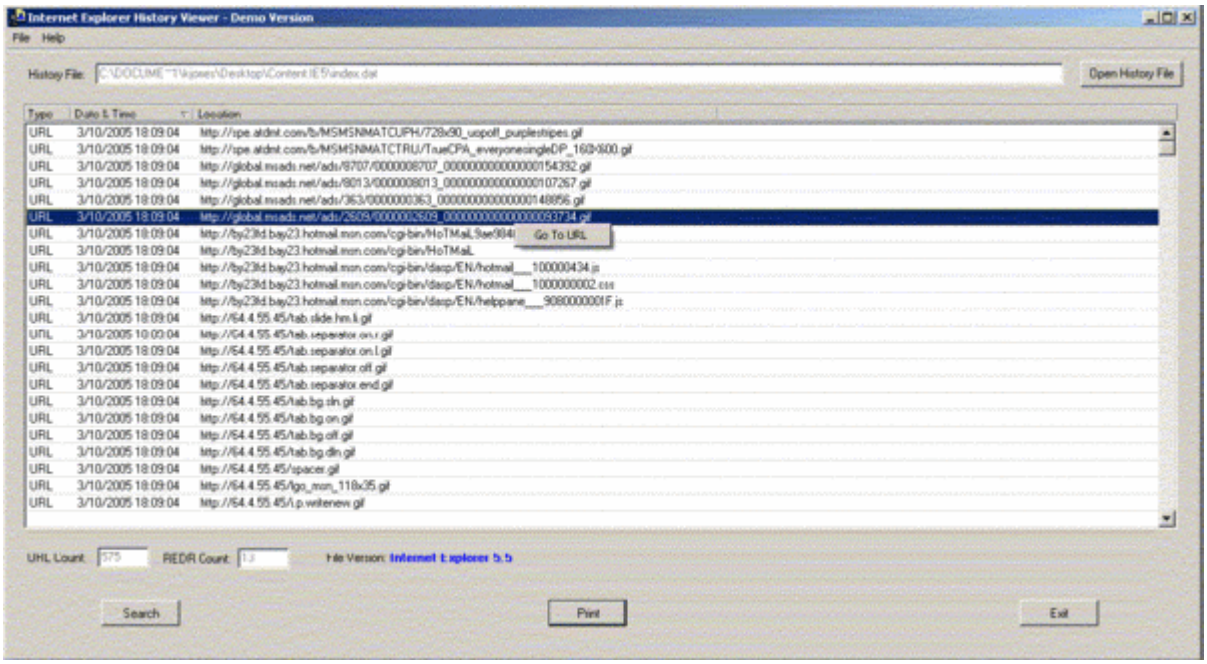


Figure 9. IE History results.

IE History is a lightweight, inexpensive tool that allows you to investigate most web based Internet activity.

Forensic Tool Kit (FTK)

FTK can be downloaded from http://www.accessdata.com/products/ftk/
The actual software cost over one thousand dollars, however, the trial version allow analyze of up to 5000 files.

FTK combines some of the functionality from all of the tools we presented in this article. As commercial tools go, this receives our highest recommendation for the ease of use alone. With FTK, you can browse the cached web pages and see them in a web browser-like interface. For example, Figure 10 shows one of the cracking sites Joe's account visited.

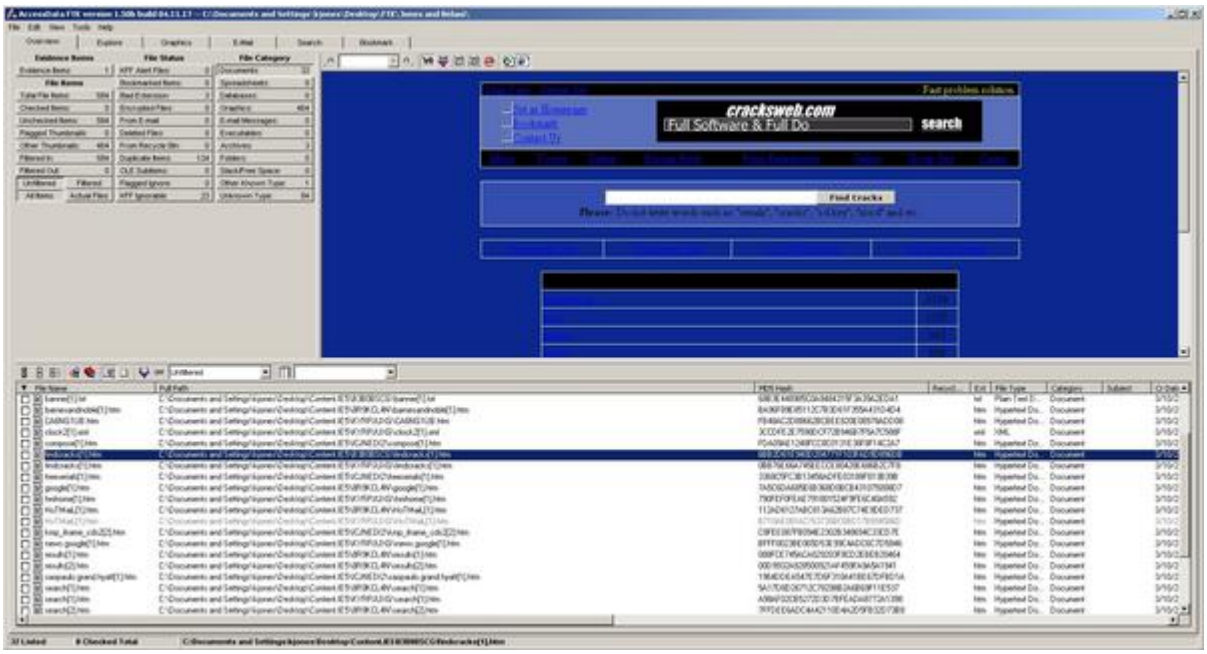


Figure 10. FTKs web activity representation.

Figure 11 and Figure 12 show that Joe was interested in Hotels in the Sao Paulo area. Since we know that Joe is currently in Florida on vacation with his family, it is highly unlikely he was the individual responsible for this activity.

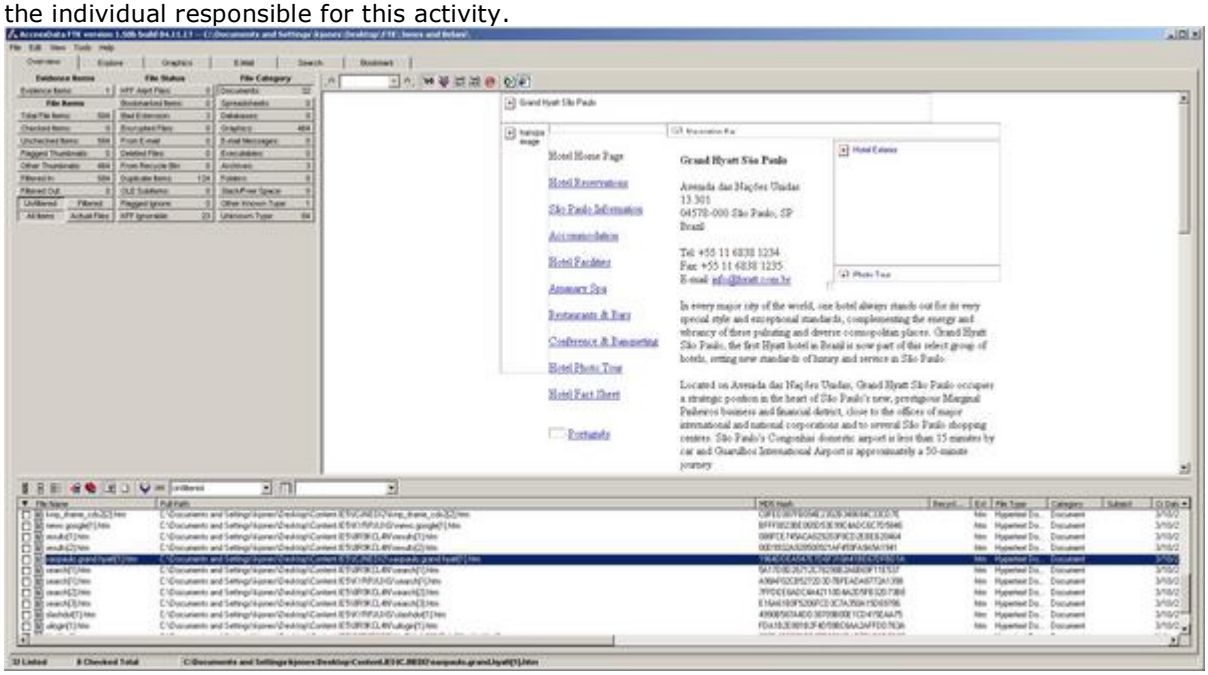

Grand Hyatt São Paulo
Hotel Home Page
Hotel Reservations
São Paulo Information
Accommodations
Hotel Facilities
Amaury Spa
Restaurants & Bars
Conference & Banqueting
Hotel Photo Tour
Hotel Fact Sheet
Portuguese
Grand Hyatt São Paulo
Avenida das Nações Unidas
13.301
04578-000 São Paulo, SP
Brasil
Tel: +55 11 6838 1234
Fax: +55 11 6838 1235
E-mail: sp@hyatt.com.br
In every major city of the world, one hotel always stands out for its very special style and exceptional standards, complementing the energy and vibrancy of these pulsating and diverse cosmopolitan places. Grand Hyatt São Paulo, the first Hyatt hotel in Brazil is now part of that select group of hotels, setting new standards of luxury and service in São Paulo.
Located on Avenida das Nações Unidas, Grand Hyatt São Paulo occupies a strategic position in the heart of São Paulo's new, prestigious Marginal Pinheiros business and financial district, close to the offices of major international and national corporations and to several São Paulo shopping centers. São Paulo's Congonhas domestic airport is less than 15 minutes by car and Guarulhos International Airport is approximately a 50-minute journey.

Figure 11. Joe's account does some travel planning.

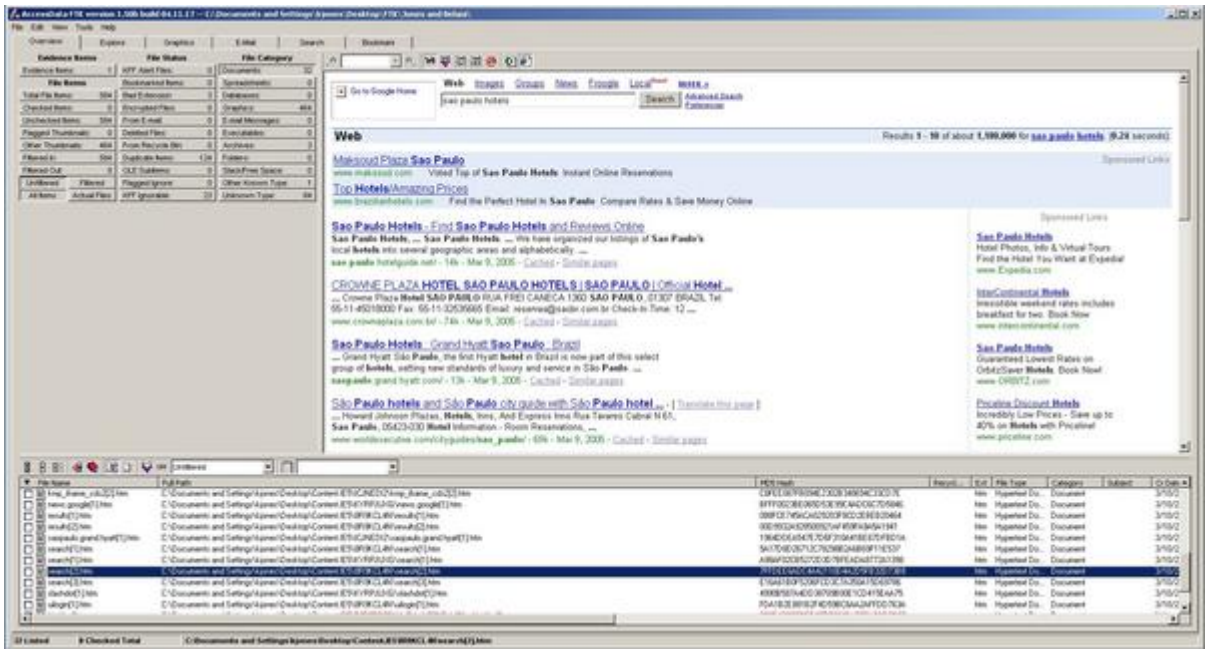


Figure 12. Joe's account was searching for hotels.

FTK reconstructs the visited web pages very well. The drawback when using FTK is the reconstruction of the `Index.dat` file. Upon selecting an `Index.dat` file within FTK, you will notice the data is presented in a format that is difficult to use. Each instance of activity is presented as a separate table, and none of the information is clickable. This means importing the data into a spreadsheet would be near impossible.

Concluding part

Our conclusion at this point is that Joe was probably not the individual using his account when unauthorized activity was performed against Docustodian. This is based on the fact that most of the potentially malicious activity occurred when Joe was on vacation with his family in Florida. However, the Internet searches for books related to hacking and license cracks are indicative of the fact that his machine may have been used by an unknown suspect. Who could it be? Who had access to Joe's machine when he was away on vacation? For the next article in this series we will examine additional investigative leads by performing an in-depth review of Joe's hard drive and the web activity of all other browsers installed on the system.

Appendix G: The Saint Jude Linux Kernel Module Intrusion Resiliency System

The Saint Jude LKM Intrusion Resiliency System provides a glimpse into an area of active research – How to make a system resilient to attacks, known and unknown, by tightening the host system towards actions an attacker may attempt in gaining root privileges on the attacked host. Current research studies how to apply this concept to virtual capable hardware and software, which would not only increase security, but allow for easy recovery in case of an attack and prevent compromising the underlying computer structure.

The Saint Jude LKM IRS attempts to tighten security in a Linux system by preventing unauthorized processes from gaining root access to the protected host through the monitoring of key Kernel functions, and only allowing those authorized processes to perform those tasks.

Introduction:

This exercise consists of installing and configuring the Saint Jude LKM IRS to protect a virtual machine from a series of attacks and attack simulations. You will need a new copy of the RedHat 7.2 Virtual machine, StJude_LKM-0.23.tar.gz, and IDSwakeup-1.0.tgz.

Background:

During recent years, significant strides have been made in the identification and elimination of software flaws that open up windows of exposure during which the integrity of host systems may be assailed. Unfortunately, the increased awareness and attention that security-related software flaws have drawn has come with a cost. The rate at which new vulnerabilities are being discovered and published is increasing at super-linear rates, according to data compiled by CERT/CC, and SecurityFocus.com (Figure 1). This growth directly translates into more frequent windows of exposure during which host systems are susceptible to being compromised.

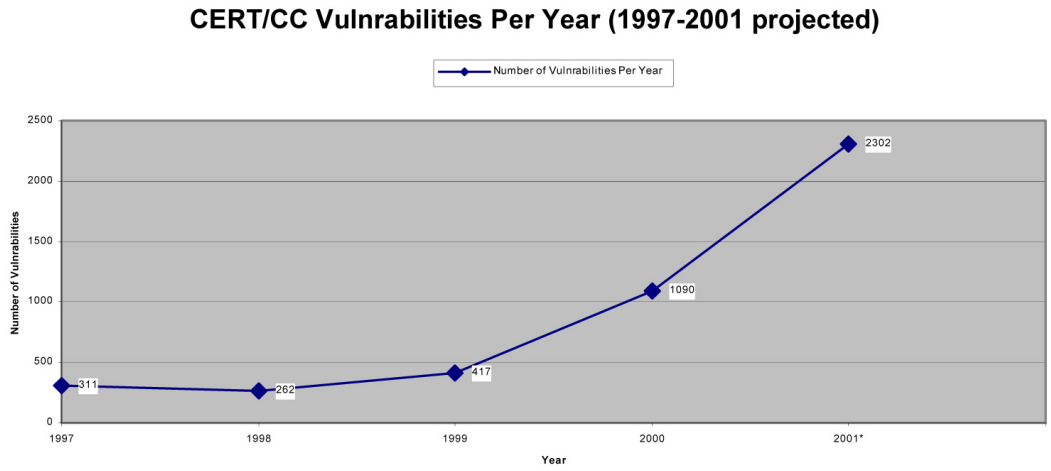


Figure 1 – Number of Vulnerabilities/Year

A Window of exposure opens for host systems of a particular class each time vulnerability is discovered. In the case of unpublished or private vulnerabilities, these windows of exposure don't readily close, and may be exploited extensively prior to the knowledge of their existence becoming public. The "threat-space" that these unpublished vulnerabilities occupy will likely expand in proportion to the space occupied by public vulnerabilities as a result of new movements within the underground to conceal vulnerabilities from public disclosure

When faced with these new, and likely unpublished, vulnerabilities -- and their associated exploits -- the various perimeter-based defensive mechanisms may offer little or no protection. Without known vulnerability information to use in populating signature databases, analysts protecting hosts systems are forced to wade through alarms generated by anomaly detectors -- weeding out the new and unknown attacks from the background noise.

Timeline of an Intrusion Incident:

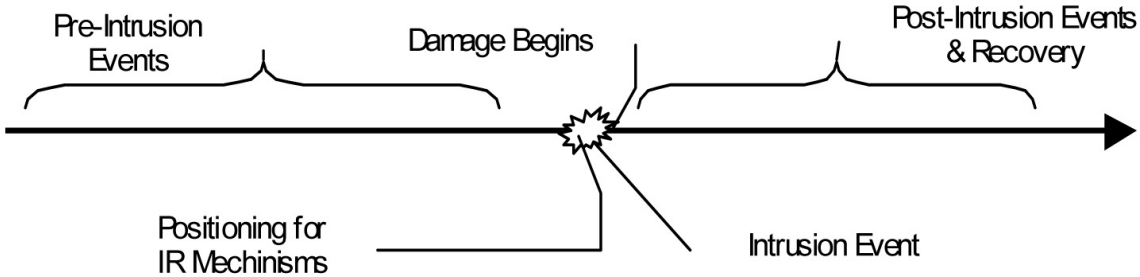


Figure 2 – Timeline of an Intrusion

The pre-intrusion event time period is spent conducting intelligence-gathering operations, staging and delivering of the attack that generates the intrusion event. Typically this is the period of time that intrusion detection mechanisms are desired to detect the pre-cursors to

the intrusion event, such as port scanning, failed authentication attempts, or file access failures.

During the pre-intrusion time span, the universe of potential sequences of events that may lead to the actual intrusion event decreases as the assailant advances towards their goal. The sequences that emerge contain behavior that may be described as a Markov Chain, with the set of possible next-steps diminishing as the assailant progresses towards their goal.

On the other end of the timeline of an intrusion, the potential for damage begins upon the conclusion of the intrusion event and continues to accumulate until such time that a response can be mounted to contain and extenuate the incident. Methods to contain this damage through compartmentalization of process-bound resources exist; however within the compartment, the intrusion is unfettered, and any trust relationships or communication channels between compartments represent possible vectors by which the intrusion may further spread – compromising additional compartments.

After containment, recovery begins where the damage that was caused is assessed and repaired. It is important to note that an intrusion, in itself, does not cause damage; it is the means by which damage latter occurs via the loss of integrity, confidentially, or availability. If it were possible to detect and neutralize an intrusion before the potential for damage presented itself, then the impact of the intrusion would be the same as if no intrusion ever occurred.

The following steps will be employed to circumvent the attacker's attempt to gain root privilege access, and thereby avoid the intrusion entirely. It is important to note that a system such as St. Jude can only work if it is entirely autonomous and independent of external action or information in order to make a decision regarding whether or not a given process in the protected host is malicious or not. Saint Jude attempts to reduce false positives and negatives by first working in a "learning" mode, when all root escalations are allowed, but information is gathered to establish the ruleset that will guide Saint Jude when in normal operation mode.

Part 1: Installing and Configuring St. Jude

This part of the exercise concentrates in simply installing and configuring the Saint Jude Linux Kernel Module first in learning mode and then later in normal mode, such that we may later test the effectiveness of this system. The following steps assume you already have duplicated (or created a fresh Red Hat 7.2 virtual machine) and downloaded the required files specified earlier in this document.

1. Type:
   #: ntsysv
2. Make sure the syslog service is enabled
3. Type:
   #: service syslog start

4. Extract the files in StJude_LKM-0.23.tar.gz by typing
   #: tar zxvf StJude_LKM-0.23.tar.gz
5. Cd into StJude_LKM-0.23
6. ./configure
7. Configure St. Jude into learning mode
8. Follow the script instructions for the next questions and confirm all default values
9. Once the configuration is complete, run make
10. Once St. Jude compiles, run
    #: cp bin/StJude.o /lib/modules/<Linux Kernel Version Number>
11. Add the following line to your /etc/inittab, right after the line that begins with "si::"
    sj::once:/sbin/insmod /lib/modules/<Linux Kernel Version Number>/StJude.o
12. Reboot the system. Do not do "init q". Reboot the system.
13. Once the system reboots, simulate a normal system operation. Ideally this system should be left running for a couple of days or so. Obviously this is not practical in this exercise, therefore we will simply run it in learning mode for a few minutes and then reconfigure St. Jude for normal operation. Be advised that this will very likely yield some "weird" system behavior such as your NIC not being recognized, or other such problems. That, however will not impede the procedure in this exercise as all the attacks outlined in this document will be run locally.

Part 2: Reconfiguring St. Jude for Normal Operation

The Following steps outline how to reconfigure Saint Jude for normal operation and to populate its rulebase. Please refer to the readme file in case of any doubts.

1. After the system has been put through a workout that mimics its normal behavior, remove the StJude module by issuing the command:
   #: /sbin/rmmod StJude
2. Collect the messages that were logged to syslog and store them in a file. It is not necessary to prune out the non-Stjude related lines, or cut the time stamps off.
3. Cd into <St. Jude uncompressed directory>/ StJude_Learning_Parser/
4. Run
   #: ./ StJude_Learning_Parser.pl filename
   where filename is the file containing the output lines.
5. This will produce a file called StJude_Rulebase__WORK.h. Edit this file. Within the file will be instructions on what modification will be necessary. Follow these directions. What you will be doing is copying over the overrides defined earlier, and linking those overrides to the head-end rule they should override to.
6. After changes are made, copy the StJude_Rulebase__WORK.h file over the StJude_Rulebase.h file located in
   <St. Jude uncompressed directory>/ StJude_LKM-0.23/includes/
   StJude_Rulebase.h
7. Rerun the configure script, only this time select the N option for normal operation

8. Make sure to confirm all default options. DO NOT INSTALL THE ST. MICHAEL MODULES AS THOSE WILL RENDER THE ST. JUDE MODULE INCOMPILABLE.
9. Recompile the module by typing:
   #: make clean
   #: make
10. Copy the StJude.o module over the StJude.o module that is refrenced by your inittab file.
11. Reboot the system
12. You may now run any of the included demo attacks, the aforementioned IDS wakeup, which simulates intrusion attacks, and any other intrusion attack you wish, including those you have already run through in previous labs, such as the Metasploit Framework attacks. In all instances you should be now denied privilege escalation and therefore the attack rendered unsuccessful.
13. To unload the module later, simply run rmmod StJude, remove the St. Jude line in /etc/inittab, and delete the StJude.o from the referenced directory. Note that you must be root to perform those actions.

References:

T. Lawless, “On Intrusion Resiliency” 12 Feb 2002.

T. Lawless, “Saint Jude, the Model” 6 Jan 2002.

ECE 4112 Internetwork Security

Lab 7: HoneyNets and Forensics

Group Number: _____

Member Names: _____ _____

Section 1: Install and run BackOfficerFriendly (A free Honeypot) on your virtual WinXP machine

Q1.1: What happens when a connection is attempted? As a network administrator, why would you want to use BOF on your honeynet?

Section 2: The Homemade Honeypot using Netcat as a Port Sniffer

Q2.1: Now look at the worm file on the Linux 7.2 machine, what do you see here?

Section 3: Set up and use Ethereal to capture packets

Q3.1: What packets did you observe?
Attachment 1: Submit screen shot of packets.

Q3.2: Describe the content of the packets.

Q3.3: What packets did you observe?
Attachment 2: Submit screen shot of packets.

Q3.4: Describe the content of the packets.

Section 4: Set up and use Snort to capture packets

Q4.1: Explain how the *–l* option organizes the logging of network traffic in your new *log* directory.

Section 5: Scan of the Month Challenge

Q5.1: What is the attacker's IP address? (Hint: You will need to convert from Hex to decimal)

Q5.2: What is the attacker doing first? What do you think is his/her motivation for doing this?

Q5.3: What is the purpose of 'foo'? Can you provide more insights about the internal workings of 'foo'? Do you think that 'foo' was coded by a good programmer or by an amateur? (Note: Eloy Paris' decompiled foo.c has been included for you.)

Q5.4: What is the purpose of './ttserve ; rm -rf /tmp/ttserve' as done by the attacker?

Q5.5: How do you think the attacker will use the results of his activity involving 'foo'?

Q5.6: If you administer a network, would you have caught such NVP backdoor communication? If yes, how? If you do not administer a network, what do you think is the best way to prevent such communication from happening and/or detect it?

Section 6: Using SNORT to act as an IDS (Intrusion Detection System)

Q6.1: Write a rule to detect the imapd-ex attack.

Q6.2: Did SNORT give an alert?
Attachment 3: Submit a screenshot or output of alert.

Section 7: Advanced uses of Ethereal

Q7.1: Specify any hacking activities from the data sets.
List details such as Time of attack (start, end and duration), Source IPs, Target IPs, Hacker's activities and Service of the target

Q7.2: Suggest possible security measures to prevent these attacks. What can be done with the information gathered in order to enhance security of the network (think firewall and IDS)?

Q7.3: Use an analysis tool of your choice to determine the following information about the attacks represented in the files *snort-0920@0001.log* and *snort-0926@1541.log*
You can do either one of them.
Specify any hacking activities from the data sets.

Enlist details such as

- **Time of attack (start, end and duration)**
- **Source IPs (IP addresses and ports)**
- **Target IPs (IP addresses and ports)**
- **Hacker's activities**
- **Hacker's objectives (were they accomplished?)**
- **OS and service of the target**

Suggest possible security measures to prevent these attacks. What can be done with the information gathered in order to enhance security of the network (think firewall and IDS)?

Section 8: Introduction to AIDE (Advanced Intrusion Detection Environment)

Q8.1: What does Aide show after each of the changes (new user and new binary file)?

Q8.2: Where is a good place to keep a clean copy of Aide and the database? Why?

Section 9: Snare for Windows

Q9.1 Identify some of the events that have already appeared. Why might this be important with regard to safety?
Attachment 4: Take a screenshot of the command window starting and point out the specific event

Take a screenshot of the command window starting and point out the specific event

Q9.2 How could this remote control functionality be useful?

Section 10: Forensics Investigation the Penguin Sleuth Kit

Q10.1: What files did the user "rpcuser" access? (Hint: grep for " rpcuser "; if you couldn't access /etc/passwd earlier, grep for " 29 " (rpcuser's UID))

How long did it take you to complete this lab? Was it an appropriate length lab?

What corrections and/or improvements do you suggest for this lab? Please be very specific and if you add new material give the exact wording and instructions you would give to future students in the new lab handout. You may cross out and edit the text of the lab on previous pages to make minor corrections/suggestions. General suggestions like add tool xyz to do more capable scanning will not be awarded extras points even if the statement is totally true. Specific text that could be cut and pasted into this lab, completed exercises, and completed solutions may be awarded additional credit. Thus if tool xyz adds a capability or additional or better learning experience for future students here is what you need to do. You should add that tool to the lab by writing new detailed lab instructions on where to get the tool, how to install it, how to run it, what exactly to do with it in our lab, example outputs, etc. You must prove with what you turn in that you actually did the lab improvement yourself. Screen shots and output hardcopy are a good way to demonstrate that you actually completed your suggested enhancements. The lab addition section must start with the form "laboratory Additions Cover Sheet" which may be found on the class web site.

Turn-in Checklist

- Answer Sheet
- Ethereal screen Shot
- Ethereal screen shot of Imapd exploit
- Screenshot of Snort rule working
- Screenshot of Snare